VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2010 Pavel Stoklasa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Monitorovací systém pro OS Linux

Monitoring system for OS Linux

2010 Pavel Stoklasa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Stoklasa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Monitorovací systém pro OS Linux
Monitoring System for OS Linux

Zásady pro vypracování:

Pokuste se navrhnout „Monitorovací systém" postavený na architektuře klient-server. Systém bude monitorovat vybrané systémové prostředky OS Linux. Systém bude obsahovat prostředí na nastavení základních parametrů aplikace na straně serveru. Data budou online publikována ve standardizovaných formátech (xml, rss) mobilním klientům (správcům), kteří pomocí inteligentních zařízení podporujících běh aplikací pro práci s RSS budou moci sledovat statistiky monitorovaných systémů.
Pokuste se navrhnout vzorovou klientskou aplikaci pro čtení formátu RSS na platformě Windows Mobile, která bude umět reagovat na nastavení prahových hodnot monitorovaného sytému. Použité technologie – PHP, BASH, JAVA, XHTML, XML, RSS, MYSQL, případně .NET – C#.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Seidl**

Datum zadání: 20.11.2009

Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: Pavel Stoklasa

Rád bych tímto poděkoval Ing. Davidu Seidlovi za odborné vedení mé bakalářské práce a také za podnětné nápady a tvůrčí invenci, se kterou se mi laskavě věnoval.

Abstrakt

Práce se věnuje vývoji monitorovací aplikace pro operační systém Linux s cílem dosáhnout vizualizaci dat na širším spektru zařízení, na kterých se budou monitorovaná data prezentovat se zvláštním zaměřením na mobilní klienty, jako jsou například chytré mobilní telefony.

V další části práce je uveden vývoj prototypu aplikace nad architekturou Windows Mobile na platformě .NET, která má za cíl přehledně vizualizovat data RSS formátu. Je zde nastíněn problém tvorby vhodného uživatelského rozhraní pro pohodlné ovládání prsty.

Klíčová slova

Tvorba monitorovací aplikace pro OS Linuxu, BASH skripty, RSS, Vizualizace textových dat, Vývoj čtečky RSS, pohodlné ovládání uživatelského rozhraní pomocí prstů

Abstract

Main work shows how to develop monitoring of applications for operation system Linux. How to reach visualization of text data from the main monitoring system in broad range of devices such as desktop computers, Internet terminals and especially on SmartPhone devices.

There is shown how to develop prototype of application for visualization of RSS formatted documents on the Windows Mobile architecture in next part of main work. There is used: .NET platform for this application building. Discussed problem is choosing a proper user interface for touch friendly control of this application.

Keywords

Developing monitoring application for OS Linux, BASH scripting, RSS, Visualize text data,
Developing RSS reader, friendly control User Interface

Seznam použitých symbolů a zkratek

.NET - „dotnet“ ucelená platforma pro vývoj aplikací

.NET CF - Compact Frameworkem, platforma pro vývoj aplikací nad mobilní architekturou WM

Apache - webový server

API - Aplication Program Interface

ATOM - Atom Syndication Format

AWK - je univerzální počítačový jazyk, navržený především pro zpracovávání textových dat

BASH - Bourne Again Shell, příkazový interpret

C# - C Sharp, vysokoúrovňový objektově orientovaný programovací jazyk

C - jde o nízkoúrovňový, kompilovaný a relativně minimalistický programovací jazyk

CAT - program vypisuje obsah souborů na standardní výstup

CDF - Channel Definition Format

CLDC - Connected Limited Device Configuration

CLR - Common Language Runtime, běhové prostředí .NET

CPU - Central Processing Unit, procesor

CRON - unixový program běžící na pozadí, provádějící opakované spouštění příkazů

Daemon - démon, programový kód dovolující běh aplikace na pozadí

Development Kit - vyvojářská sada nástrojů

DF - unixový nástroj zobrazující využití diskových oddílů

DML - Data Manipulation Language

DOM - Document Object Model

Eclipse - vývojové prostředí

ESX - virtualizační platforma firmy VMware

GNU GPL - General Public License, všeobecná veřejná licence

Google - název celosvětového vyhledávače

GSM - globální systém pro mobilní komunikaci

HotFix - opravný software

HTML - Hyper Text Markup Language

HTTP - Hypertext Transfer Protocol, internetový protokol pro přenos hypertextových dokumentů

HTTPS - Hypertext Transfer Protocol, to samé co HTTP obohacený o SSL vrstvu

I/O - Input/Output

IP - Internet Protocol

iPhone OS - operační systém od společnosti Apple pohánějící přístroje iPhone a iPad

IDE - Integrated Development Environment

IT - Information Technology, informační technologie

Java - objektově orientovaný programovací jazyk

Java ME - Java Platform Micro Edition

JVM - Java Virtual Machine, virtuální stroj javy

Kernel - jádro operačního systému

LAMP - Linux, Apache, Mysql, PHP

LDAP - Lightweight Directory Access Protocol, protokol adresářových služeb

LINQ - Language Integrated Query

Linux - operační systém

MRTG - monitorovací aplikace

Mysql - systém řízení báze dat

Nagios - monitorovací aplikace

NetBeans - vývojové prostředí od společnosti Sun

OS - Operation System, operační systém

OOS - Open-source, programy s otevřeným zdrojovým kódem

PDA - Personal Data Administrator

PHP - Profesional Homepage, skriptovací jazyk

Pixel - Picture Element (px), obrazový prvek

PostgreSQL - systém řízení báze dat

Proc - virtuální adresář

PS - unixový program, který zobrazí přehledně procesy systému

Pull parser - druh API zpracovávajícího XML dokument po částech

RAM - Random Access Memory, operační paměť počítače

RDF - Resource Description Framework

RecordStor - datové úložiště Java ME

RSS - Really Simple Syndication

Rsync - program zajišťující na unixových systémech synchronizaci souborové hierarchie

SCP - Secure Copy, označení jak pro protokol, tak pro aplikaci využívající tento protokol

SFTP - SSH File Transfer Protocol, protokol a aplikace pro zabezpečený přenos souborů

SIP - Software Input Panel

SmartPhone - chytré zařízení kapesních rozměrů, většinou s bezdrátovým připojením k Internetu

SNMP - Simple Network Management Protocol

SP - Service Pack

SQL CE - Microsoft SQL Server Compact, SQL Server Mobile Edition

SQLite - relační databázový systém

SŘBD - Systém Řízení Báze Dat

SSH - Secure Shel, jde o program využívající zabezpečený protokol pro komunikaci mezi serverem a klientem

SSL - Secure Sockets Layer, bezpečnostní vrstva vložená mezi vrstvu transportní

Stylus - ovládací prvek dotykových zařízení většinou ve tvaru tužky (dnes pomalu na ústupu)

SWAP - odkládací oddíl, který používá OS Linux při nedostatku paměti RAM

Sun - Sun Microsystems, Inc.

Ubuntu - název distribuce operačního systému Linux

UI - User Interface, uživatelské rozhraní

Unix - druh operačních systémů

URL - Uniform Resource Locator, jednotný identifikátor umístění zdroje (např. dokumentu)

VS - Visual Studio 2008, vývojové prostředí od společnosti Microsoft pro platformu .NET

W3C - World Wide Web Consortium, mezinárodní konsorcium vyvíjející webové standardy

WiFi - Wireless Fidelity, standard pro lokální bezdrátové sítě

WYSIWYG - „What you see is what you get“, styl vizuálního návrhu, např. u webových dokumentů

WM - Windows Mobile, mobilní architektura firmy Microsoft

WMI - Windows Management Instrumentation

XML - Extensible Markup Language

xHTML - Extensible Hypertext Markup Langure, značkovací jazyk vyvinutý konsorciem W3C

Zabbix - monitorovací aplikace

Obsah

1 Úvod

Proč vlastně monitorovat nějaký systém? K čemu může být dobré vědět o stavu systému a jak může být těchto informací využito? Pomocí jakých prostředků lze efektivně systémy monitorovat?

Hned na úvod je zapotřebí zamyslet se nad tím, proč vlastně monitorovat operační systémy, hardware na kterém běží a jejich celkovou vitálnost. Také bych se rád zamyslel nad tím, jaké výhody a úskalí daný problém s sebou nese, jakých cílů a za pomocí kterých technologií jich dosáhnout.

Již od počátku, kdy dostupnost serverových služeb začala být jednou ze zásadních veličin poměřování kvality poskytnutých služeb systémem, bylo jasné, že bude potřeba sledovat chování systému. Tento fakt přispěl k tomu, že především na straně serverových operačních systému se začalo provádět sledování neboli monitorování stavu daného systému.

Monitorování serverových operačních systémů je jednou ze základních činností systémového administrátora nebo IT odborníků zodpovědných za chod a dostupnost prostředků sítě. V dnešní době existuje mnoho komplexních nástrojů, které jsou schopny sledovat a monitorovat rozsáhlé intranetové sítě a jejich prostředky, a to vše z jednoho místa. Tak mohou přispět k efektivnímu chodu síťového prostředí a k napomáhání s odhalováním slabých míst a zajistit vysokou dostupnost monitorovaných systémů. Tyto nástroje jsou velice komplexní a robustní, většinou se zakládají na architektuře klient - server. Jsou schopny monitorovat síťová zařízení jako například: počet přenesených dat, stav zatížení systému a další statistiky. Na vyšších vrstvách tyto komplexní monitorovací nástroje sledují a sbírají informace o serverech a pracovních stanicích. Na serverech se hlavně monitorují výkonnostní parametry, aby mohla být odhalována slabá místa v systému. Sleduje se nejčastěji výkon serverového systému, jako například zatížení procesoru (CPU), využití paměti (RAM), dostupné místo na disku, počet vstupně/výstupních operací (I/O) atd. Mimo tyto údaje jsou sledovány také konkrétní služby (aplikace) a hlavně jejich dostupnost v systému, jako příklad můžou být: webový server, systém řízení báze dat (SŘBD), souborový server, adresářové služby (LDAP), atd. U těchto služeb je sledována především jejich dostupnost. Systém monitorující takovouto službu může v případě výpadku služby vhodně zareagovat - restartováním služby, posláním výstražného e-mailu správci serveru, atd. Tato práce si neklade za cíl zkoumání funkčnosti jednotlivých monitorovacích systému.

Jak bylo nastíněno výše v textu - monitorování vychází z potřeby mít přehled o systému, o jeho subsystémech, ať už se jedná o aplikační část nebo hardwarovou část nebo jde o celkovou vitalitu systému. Většina těchto informací je přístupná přes aplikační rozhraní operačního systému. Například v případě Microsoft Windows jde o WMI (Windows Management Instrumentation). Na platformě operačního systému Linux lze informace o systému zjistit z virtuálního adresáře „*./proc*“. Tento adresář je vytvořen jádrem systému (kernelem) při startu systému. V adresáři /proc je umístěna adresářová struktura, která reprezentuje nastavení a stav operačního systému. Drtivá většina informací je zde reprezentována v textové podobě. Pro tento snadný a účinný přístup k datům byl vybrán právě operační systém Linux, jako vzorový případ sledování systému.

Operační systém Linux vychází z kořenů výkonných síťových unixových systémů. V dnešní době je tento systém hojně využíván pro různá nasazení, ze kterých se nejčastěji používají například tyto: webový server - Apache, systém řízení báze dat - Mysql a skriptovací jazyk - PHP. Pro tuto kombinaci

serverových aplikací se již vžil název LAMP (Linux, Apache, Mysql, PHP). Nejen tyto aplikace přispívají k veliké oblibě operačního systému Linux, ale i spousta dalšího svobodného softwaru, který je zdarma dostupný, tak jako operační systém sám. Jádro systému je pro všechny distribuce Linuxu stejné, jen se distribuce liší svou podporou a dostupným programovým vybavením. Je dále nutno podotknout, že operační systém Linux si získává svou oblibu i na koncových stanicích běžných uživatelů, kde se pomalu zabydluje. Jako příklad může být uvedeno jméno distribuce Linuxu, které je již synonymem pro Linux samotný - Ubuntu. Fenomén této distribuce dokládá i to, že v celosvětovém gigantickém vyhledávači Google začíná vyhledávané slovo „Ubuntu“ předbíhat vyhledávané slovo „Linux“.

Tato distribuce je vhodnou platformou pro nasazení monitorování a vývoj monitorovacího systému a to hlavně z hlediska dostupného programového vybavení, jednoduchosti správy a pro širokou podporu této vyzrálé distribuce. Nicméně, nic by nemělo ubírat z obecných principů zde popsaných řešení, které by měly být přenositelné i na jiné distribuce Linuxu. Dále bude použito aplikačního vybavení distribuce Ubuntu k vybudování menšího monitorovacího systému, který bude sloužit jako demonstrace snadného nasazení monitorování operačního systému založeného na architektuře klient - server.

Na straně serverového operačního systému se nabízí použití již zmíněného LAMP pro zajištění sbírání, zpracování a zobrazení dat. Data o systému budou shromažďována a zpracovávána pomocí kombinací skriptovacích jazyků BASH a PHP. Data budou uložena v SŘBD a následně pomocí protokolů HTTP či HTTPS poskytována ve formě RSS dokumentů klientským prohlížečům.

Formát RSS je dnes široce používán k syndikaci obsahu webových serverů, které poskytují informace různého druhu. Použití RSS jako formy předávání zpráv o stavu systému je jistě neotřelým nápadem. Dnes existuje spousta klientského programového vybavení, které je schopno formát RSS přehledně zobrazit. Hlavní síla se ukáže ve chvíli, kdy klientská aplikace pro sledování RSS dokumentů běží na mobilním zařízení jako například na PDA (Personal Data Administrator) či SmartPhone, nebo jiných chytrých internetových terminálech, které obsahují podporu pro zobrazení RSS dokumentu. Mnoho IT specialistů je již obdobným „chytrým“ mobilním zařízením vybavena a s klesající cenou za datové tarify je přímou výzvou ke sledování vzdáleného systému pomocí technologie RSS a SmartPhone.

Vlastní prototyp klientské aplikace pro sledování systému bude popsán a vybudován v prostředí .NET CF a to konkrétně v jazyce C#.

Jako ideální platforma pro nasazení a vývoj vizualizéru RSS dokumentů se jeví platforma WM (Windows Mobile) v čele s .NET technologií, přesněji .NET Compact Frameworkem (CF), jež je odvozena z většího frameworku .NET. Technologie .NET CF pochází z dílny firmy Microsoft. Mnoho společností dnes využívá platformy Windows Mobile spolu s .NET CF pro vývoj svých interních informačních systémů a systému pro podpůrné agendy nezbytné pro chod firmy. Kvalitní dokumentace, dostupnost vyspělého vývojového prostředí spolu s emulátorem platformy WM a neméně dobré programovací jazyky z platformy .NET, mezi které jistě patří C#, Visual Basic atd., dělají vhodnou základnu pro vývoj aplikací.

Konkurenční technologie od společnosti Sun se jmenuje Java ME. Jak plyne z názvu, půjde o odvozeninu architektury založené na jazyku Java. Úroveň implementace virtuálního stroje java (JVM)

a jeho profilů (CLDC) je u různých zařízení bohužel dost odlišná. Tato odlišnost je daní za to, že JVM běží na spoustě zařízení, které se mnohdy razantně liší svou hardwarovou výbavou, jako např.: velikostí operační paměti, rychlostí procesoru, velikostí a barevnou hloubkou zobrazovací jednotky. Tyto negativní aspekty platformy Java ME ji trochu znevýhodňují před technologií .NET CF.

2 Monitorovací systém pro OS Linux

Pro platformu operačního systému Linux existuje několik open source (OOS) monitorovacích řešení, které jsou z větší části postavené na architektuře klient - server. Mezi ty známější jistě patří Nagios [1], Zabbix [2], MRTG [3] a další. Všechny výše zmíněné aplikace jsou již komplexní nástroje obsahující spoustu funkcionalit, ale u všech jsem postrádal export dat do formátu RSS tak, abych je mohl kdekoliv, kde je dostupné připojení k Internetu a za pomoci mého mobilního terminálu interpretovat a zjišťovat stav serveru na dálku. Na následujících stránkách bude ukázán vývoj a nasazení jednoduché monitorovací aplikace v prostředí operačního systému Linux, která bude monitorovat stav operačního systému a bude jej prezentovat ve formě RSS (viz. Příloha A) a xHTML dokumentů.

2.1 Návrh řešení monitorovacího systému

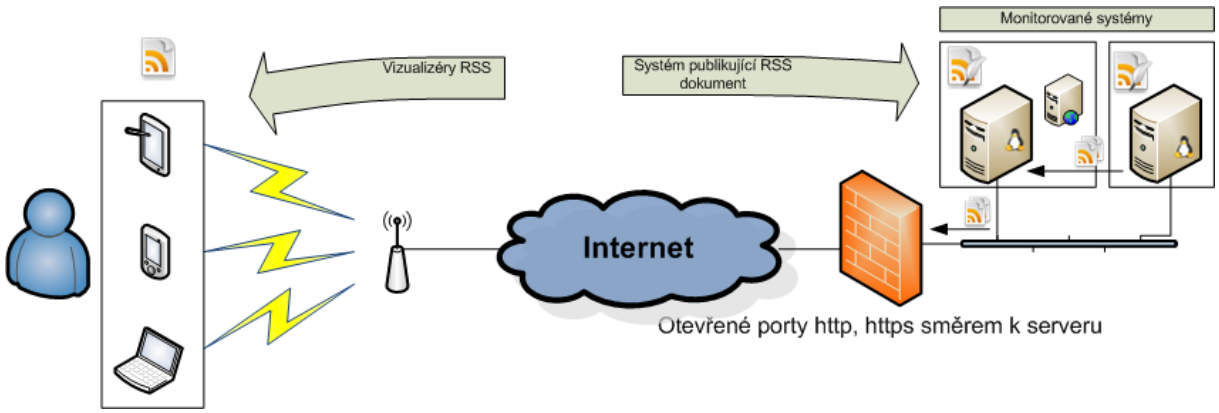
Zásadnější otázkou bude získání relevantních informací o stavu systému tak, aby operační systém nebyl zbytečně zatěžován a také aby budoucí vývoj monitorovacího systému nebyl příliš složitý a implementačně náročný v případě nasezení v reálném provozu.

Hlavním cílem této práce bude implementace monitorovacího systému, který bude sloužit pro sledování OS Linux za použití dostupných nástrojů přítomných přímo v systému. Z tohohle pohledu budou vycházet požadavky na všechny použité technologie při implementaci monitorovacího nástroje.

2.1.1 Architektura monitorovacího systému

Monitorovací systém bude založen na již zmiňované architektuře klient - server. Serverová část monitorovacího nástroje bude shromažďovat informace o stavu systému. Dále bude zajišťovat mimo jiné taky:

- Persistenci shromážděných dat po kratší časovou periodu
- Prezentaci uložených dat na co možná největším počtu klientských zařízení, se zvláštním zaměřením na chytré mobilní zařízení
- Snadná a dostupná komunikace s klientem



Obrázek 1

Obrázek 1 zobrazuje návrh řešení architektury monitorovacího systému.

2.1.2 Shromažďování informací o stavu systému - monitoring

Na platformě OS Linux existuje několik možností, jak získat informace o stavu systému. Například může jít o speciálně napsanou aplikaci v jazyce C nebo může jít o modul jádra systému. Existují také nádstavby a rozšíření OS o sadu nástrojů a protokolů, jako například SNMP, který je schopen se na dálku po síti dotazovat na stav systému, kde je nainstalován a konfigurován (za určitých podmínek lze SNMP protokol využít i k nastavování parametrů systému po síti). SNMP protokol se používá většinou jako komunikační protokol v komplexních monitorovacích nástrojích, protože jde o standardizovaný protokol z rodiny IP a výrobci různých síťových zařízení jej implementují jako jednu z přidaných hodnot dohledu nad svým zařízením. V současné době existují 3 verze protokolu SNMP. Největší oblibě se těší verze druhá, která přidává autorizační mezivrstvu do implementace SNMP protokolu. SNMP protokol je vhodný pro sledování prostředků větší sítě, ale pro účely tvorby menší monitorovací aplikace je příliš komplexní a robustní.

Další možností jak sledovat informace spojené se stavem OS v prostředí Linuxu je virtuální adresář „*/proc*“. Adresář /proc je virtuální souborový systém, který je vytvořen jádrem operačního systému Linux po každém startu systému. Jádro systému - kernel zde intepretuje svůj stav a nastavení a to většinou ve formě textových informací. Jsou zde k mání informace o přítomném hardware počítače, který byl kernelem identifikován a zprovozněn. V adresáři /proc kernel také shromažďuje statistiku využití operační paměti, zátěž procesoru a mnoho dalších cenných informací. Skrze /proc adresář může být ovlivněn i běh samotného kernelu, a proto přístup k tomuto adresáři by měl být citlivě posouzen ze strany administrátora systému.

Informace poskytnuté virtuálním adresářem /proc jsou na dostatečné úrovni a lze k nim snadno přistupovat bez implementace další mezivrstvy. Z pohledu efektivity získávání a jednoduchosti zpracování informací je adresář /proc vhodným kandidátem jako zdroj informací o aktuálním stavu monitorovaného OS. V některých případech je vhodnější sáhnout po speciálně napsaných unixových nástrojích, které jsou schopny přehledněji zobrazit informace z /proc adresáře a neprovádět tak pársování nad stovkami adresářů či souborů, což by mělo za následek razantní snížení výkonu aplikace a ovlivnění naměřených hodnot. Použití speciálně napsaných nástrojů pro určité oblasti informací z adresáře /proc rozhodně nijak nezpomalí monitorovací aplikaci, protože drtivá většina těchto nástrojů je psána v jazyce C a přesně na míru svému specifickému úkolu.

2.1.3 Rozsah monitorovaných dat

V budoucí aplikaci nebudou sledovány úplně všechny informace o systému. Budou vybrány pouze konkrétní partie, které jsou relevantní pro systémového administrátora, a ze kterých se dá usuzovat stav systému. Mezi jednotlivé části např. patří:

- Využití procesoru (CPU) [%]
- Celkové, volné a obsazené místo na disku [B]
- Celková, dostupná a využitá paměť systému (RAM + odkládací oddíl) [B]
- Celkový počet přenesených bajtů síťovým rozhraním [B]
- Využití systému [%]
- Počet spuštěných procesů [-]
- Doba běhu systému [Dny]

Z některých údajů půjde vypočítat průměrný stav sledované části, například za 1 hodinu. Konkrétní údaje budou zakomponovány při samotné tvorbě aplikace a budou poplatné každé sledované části systému.

2.1.4 Programové vybavení

Možné zdroje informací o systému byly popsány výše v textu a nyní nadešel čas se zamyslet nad tím, jaké techniky použít při získávání dat. Vhodné aplikační řešení se bude opírat o nástroje, které jsou dostupné napříč unixovými systémy a které prošly dlouholetým vývojem. Na základě tohoto pohledu se zůží výběr nástrojů, programovacích jazyků a dalších součástí potřebných pro vývoj budoucí aplikace. Dalšími omezujícími kritérii pro volbu vhodných nástrojů může být bezpečnostní riziko, vliv na výkon systému a mnoho dalších atributů.

Kupříkladu, ne každý administrátor ochotně bude instalovat jazyky Perl nebo PHP na server, aby pomocí těchto jazyků zprovoznil monitorovací aplikaci. Na některých systémech, kde je bezpečnost hlavním požadavkem, je nemyslitelné, aby zde byly instalovány jakékoliv potencionálně zneužitelné aplikace.

Programové vybavení (vybraný programovací jazyk) musí podporovat běh opakovaných příkazů tak, aby bylo možno programově spouštět zpracování dat z /proc. Další podporovanou vlastností by měla být podpora běhu na pozadí tak, aby uživatel, pod kterým aplikace bude spuštěna, nemusel být přihlášen a v jeho instanci sezení by musela aplikace nepřetržitě běžet.

Po zvážení všech požadavků kladených na budoucí aplikaci jsem se přiklonil k příkazovému interpretu BASH [4], který je dostupný na drtivé většině unixových systémů včetně Linuxu. Dalo by se namítat, že se jedná o pouhý interpret příkazů, který bude pomalý a těžkopádný pro psaní takovéto aplikace. Určitě se dá souhlasit s tím, že nepůjde o výkonnostní špičku jako například zmiňovaný jazyk C. BASH má mnoho jiných kladných rysů, pro které je velice ceněn, a nezřídka je k nalezení například skript psaný v BASH, který je schopný pracovat jako HTTP server.

Pro potřeby menší aplikace, která bude vykonávat opakované příkazy, menší výpočetní operace, pársovat výstupy z textových souborů nebo unixových příkazů, je naopak BASH neocenitelným pomocníkem. Generování dat v podobě XML/RSS struktury pro něj bude hračkou. A hlavní deviza byla zmíněna výše - všudypřítomnost. Dalším pomocníkem vhodným pro zpracovávání textu tam, kde si neporadí samotný BASH, je jazyk AWK [5]. Přítomnost jazyka AWK není považována za bezpečnostní hrozbu a také jej lze označit za standard. Opakované spuštění BASH skriptů se dá zajistit buď pomocí démona CRON, nebo naprogramováním BASH skriptu tak, aby běžel jako démon proces na pozadí a v rámci této instance periodicky spouštěl potřebné příkazy, anebo kombinací obou předchozích možností.

2.1.5 Persistence monitorovaných dat

Aplikace, která bude sledovat hodnoty, bude muset také řešit ukládání těchto dat pro další zpracování, jako například výpočty nad získanými daty, vracení se v historii a vizualizaci. Persistenční vrstva monitorovací aplikace musí zajistit hladký přístup k datům tak, aby se nestala úzkým hrdlem aplikace. Na poli OS Linuxu může být použito různých technik k persistenci dat. Například ukládání dat do textových souborů, XML struktura a SŘBD. Poslední jmenovaný zástupce persistenční vrstvy bude vhodným kandidátem pro ukládání většího počtu obdobných dat. Provádění výpočtů nad daty v SŘBD

pomocí klasických funkcí jazyka SQL bude také bezproblémové. K uloženým datům se dají přidávat i další metadata. Obecně lze říci, že zpracování dat pomocí SŘBD bude pohodlné a rychlé.

Do úvah o vhodném výběru SŘBD zasahují také bezpečnostní faktory a dostupnost na cílovém OS. Teď je správná chvíle vybrat konkrétní implementaci SŘBD. Mezi známější OOS zástupce SŘBD lze jmenovat Mysql, PostgreSQL a SQLite. Mysql je vyspělý serverový databázový systém a to samé lze říci i o PostrgreSQL. Oba tyto SŘBD jsou již příliš velké a mohutné pro zámyšlenou aplikaci. Oba jsou bez problémů dostupné ve formě zdrojových kódů nebo v podobě binárních instalačních balíčků. U obou se využívá serverový proces, který se stará o činnost SŘBD. Poslední z trojice jmenovaných SŘBD je SQLite. Jedná se o minimalistický [6] SŘBD, který nepoužívá serverového démona a jde pouze o malý program, který ukládá data buď do paměti, nebo do souborového systému ve formě souboru.

SQLite je přítomný na valné většině OS včetně Linuxu. Snadno se spravuje a navíc implementuje většinu standartu SQL – 92 [6], který je pro monitorovací aplikaci více než dostačující. SQLite obsahuje standardní datové typy známé z vyspělejších SŘBD. Dobrá dostupnost SQLite, nulové riziko potencionálního nebezpečí, žádný serverový proces zodpovědný za běh SŘBD plus dostupná podpora v majoritních jazycích z něj dělá výborného kandidáta pro nasazení spolu s BASH skripty, které za pomoci volání patřičné aplikace budou spravovat databázové soubory.

U aplikace menšího rozsahu, jako je tato zamýšlená, nebudou ovlivňovat nepříznivě faktory jako je výkon zvoleného SŘBD či masivní paralelismus nad uloženými daty.

SŘBD bude uchovávat shromážděná data v krátkodobém horizontu cca několika dnů. Po uplynutí časové konstanty budou shromážděná data postupně mazána a nahrazována daty novými. Takto bude zajištěn efektivní chod systému, kdy uživatel aplikace nebude muset ručně provádět údržbu SŘBD.

Datový slovník je navržen na základě kapitoly 2.1.3.

Tabulka 1

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
modul	tstamp	Timestamp	-	N	N	-	časové razítko
	value	Integer	-	N	Y	-	naměřená hodnota

Konkrétní údaje názvu entit a počtu atributů budou stanoveny při samotném vývoji aplikace. Databázová vrstva aplikace nebude obsahovat složitější schéma, a proto zde nebude uvedeno konceptuální schéma.

2.1.6 Prezentace monitorovaných dat

Uložená data ze SŘBD bude nutno patřičně transformovat do podoby, která bude vhodná pro zobrazení na zařízeních s malou zobrazovací jednotkou cca 320 x 240 zobrazených bodů, a také na klasických počítačích. Tato množina zařízení je velice nekoherentní a těžko se bude hledat společný průnik vhodných technik pro zobrazení pořízených dat.

Jedna z možností, jak vizualizovat data uložená v databázi, je použít čistý textový formát. Klasický text bude zobrazitelný na drtivé většině zařízení, tak jak byly nastíněny výše. Dnešní doba si ovšem žádá sofistikovanější přístup k datům než je jen „obyčejný“ text. Existují různé čtečky, které jsou dostupné díky podpoře konkrétního formátu dat napříč různými zařízeními, od chytrých mobilů počínaje a u klasických počítačů konče. Tyto čtečky „vizualizéry“ dokáží transformovat strukturovaný dokument do podoby vhodné pro zobrazení na daném zařízení. Tyto čtečky mají také další schopnosti jak ulehčit uživateli práci s prohlížením informací. Například mohou stahovat informace v určitých intervalech, prohledávat data, třídit data na základě klíčových slov, atd.

Jeden z velice populárních formátů pro distribuci informací se stal formát RSS (viz. Příloha A). Dokumentů RSS se převážně využívá k syndikaci obsahu webových stránek, které poskytují jakékoliv informace (například informace ze světa, sportu, kultury, atd.). RSS [7] vychází z rodiny XML dokumentů, který strukturuje data v něm obsažená. Popularita RSS technologie je dnes obrovská. Snad každý moderní web poskytující dynamicky měnící se obsah také nabízí šíření informací právě pomocí RSS technologie. Na tuto oblibu také reagují výrobci software, kteří produkují čtečky dokumentu RSS na různé platformy. Integrace jde tak daleko, že nejoblíbenější prohlížeče webových stránek jako například Mozilla Firefox či Internet Explorer mají zaintegrovanou nativní podporu pro zobrazení RSS dokumentů.

U RSS dokumentů jde o nespornou výhodu v podobě nativního prohlížeče přítomného na většině dnes používaných OS. Podpora pro zobrazení RSS dokumentů je uspokojivá i na mobilních platformách. V kapitole 3 této práce bude předveden vývoj prototypu RSS čtečky [8] pro zařízení nad architekturou WM.

Flexibilita RSS formátu je obrovská a lze ji demonstrovat například tím, že ji lze využít pro vizualizaci dat pořízených monitorovací aplikací.

2.1.7 Distribuce dat klientům

Většinou jsou RSS dokumenty publikovány pomocí webových serverů, odkud si je klientské aplikace mohou stáhnout. Pro samotný transport dat je použit HTTP nebo HTTPS protokol. U obou protokolů jde o léty prověřené standardy. Oba protokoly nejsou často filtrovány paketovými filtry a jsou nejčastěji používány právě pro přenos RSS dokumentů. Na straně novějších klientů je implementace podpory HTTP či HTTPS na dobré úrovni a lze se také opřít o přítomnost těchto standardních protokolů v OS.

Monitorovací aplikace sestaví RSS dokument, který následně vypublikuje do prostoru webového serveru, který se postará o jeho distribuci přes síť směrem ke klientským zařízením. V případě, kdy aplikace na monitorování bude nasazena na vzdáleném systému bez přítomnosti webového serveru, zajistí pro generovaný RSS dokument přenos na jeden centrální webový server, který bude zodpovědný za distribuci dat klientům. Tento transport může být zajištěn například pomocí SCP[1], SFTP[2] protokolů nebo pomocí programu Rsync. Ve své podstatě se jedná o zabezpečený přenos souborů a bude postačovat pro přenos dat z monitorovaného systému na systém s webovým serverem. Při použití aplikací SCP nebo SFTP je jedna velká výhoda užití, a to je použití autorizace pomocí

[1] SCP patří do rodiny protokolů SSH.
[2] SFTP patří do rodiny protokolů SSH.

veřejných klíčů. Heslo ke vzdálenému serveru není na klientské straně uloženo v podobě čistého textu. V případě narušení bezpečnosti systému získá útočník pouze přístup k tomuto klíči a nikoliv k heslu, které má uživatel nastaveno napříč systémy.

2.2 Realizace monitorovacího systému

Jak bylo uvedeno v kapitole 2.1.4, bude pro vybudování budoucí aplikace využito nativních programovacích jazyků a technik, které jsou již v základu přítomny v operačním systému Linux. Půjde o skripty psané pro příkazový interpret BASH, dále v textu také označovány jako programový kód či programovací jazyk. Také budou sledovány pouze dynamické změny systému, které jsou popsány v kapitole 2.1.3.

Vývoj bude popsán na distribuci Ubuntu 8.04 LTS[3]. Zde uvede postupy a nasazení budoucí aplikace v prostředí jiných distribucí systému Linux bude totožné nebo se bude lišit jen v mírných detailech, které nebudou ubírat na obecnosti zde uvedených programovacích praktik.

Název projektu byl zvolen z hlavních oblastí užití: sys, info, rss - „sysinfo2rss„

2.2.1 Členění aplikace - struktura projektu

Aplikace se nebude skládat pouze z jedné části, tedy jednoho samostatného programu, ale půjde o několik částí programového kódu, které budou tvořit jeden celek. Tato modularita bude výhodná při oživování a ladění celého projektu.

V předchozích kapitolách lze identifikovat tyto jednotlivé části:

1. Moduly obsluhující sběr dat o systému + parsování dat a uložení naměřených hodnot do SŘBD
2. Démon, který bude v pravidelných intervalech spouštět moduly zodpovědné za sběr dat
3. Programový kód zodpovědný za výpočty a přenos dat do hlavní databáze
4. Údržba dat uložených v SŘBD
5. Generování RSS dokumentu z dat uložených v SŘBD
6. Publikovaní RSS dokumentu do cílového umístění (lokální či vzdálené)
7. Pomocný kód pro vytvoření databázových souborů, tabulek a kód kontrolující správnost nastavení důležitých hodnot pro chod aplikace
8. PHP skripty pro zobrazení detailu článků RSS dokumentu

2.2.1.1 Adresářová struktura a umistění jednotlivých skriptů

```
 1:  sysinfo2rss/         # kořenová složka projektu
 2:  |-- INSTALL.TXT      # popis instalace aplikace
 3:  |-- README.TXT       # popis aplikace, skriptů
 4:  |-- db
 5:  |   `-- sysinfo2rss.db       # databáze aplikace
 6:  |-- modules # adreář s umístěním modulů, struktůra je obdobná pro všechny
moduly
 7:  |   |-- cpu # kořenová složka modulu
 8:  |   |   |-- db
 9:  |   |   |   `-- cpu.db       # databáze modulu
10:  |   |   `-- scripts  # adresář se skripty
```

[3] Jde o serverovou distribuci Linuxu s dlouholetou dobou podpory.

```
11:  |   |       |-- cpu.sh       # skript zjistí využití CPU a uloží získaná data
do databáze
12:  |   |       |-- manage-db.sh # skript vyvtoří, smaže databázi modulu
13:  |   |       `-- var.sh       # soubor s lokálnimi proměnými modulu
14:  |   |-- disk
15:  |   |-- mem
16:  |   |-- net
17:  |   `-- system
18:  |-- scripts
19:  |   |-- web-php # zde jsou uloženy skripty pro zobrazení datailu RSS článku,
skripty se musí nahrát na web server a patřičně nakonfigurovat
20:  |   |-- dump-mudules-data-to-db.sh    # skript slouží pro získání a parsování
dat, které uloží do hlavní databáze aplikace
21:  |   |-- maintan-db.sh        # skript odstraňuje staré záznamy ze všech
databází, implicitně se odebírají záznamy starší než 1 den
22:  |   |-- mainvar.sh   # globalní soubor s proměnými v celé aplikaci
23:  |   |-- make-rss.sh  # skript na základě dat z hlavní databáze sestaví xml
soubor
24:  |   |-- manage-db.sh # skript spouští všechny manage-db.sh modulů
25:  |   |-- run-modules.sh       # skript spustí všechny moduly aplikace
26:  |   |-- sysinfo2rss-daemon.sh        # démon skript, který řídí spouštění
jednotlivých skriptu a loguje události do logovacího souboru
27:  |   |-- system-compatibility-check.sh # skript zjisťuje přítomnost nástrojů
potřebných pro chod aplikace, schopnost číst data z /proc a přípojné body svazků
28:  |   `-- upload-rss-to-web-server.sh   # skript zajistí po nastavení nahrání dat
na server.
29:  `-- tmp     # adresář obsahuje soubor s pid démon skriptu, logovací soubor,
sestavený Rss dokument
```

Adresář modules a jeho podsložky (řádky 6 - 17) je obdobný pro všechny moduly a výše je uveden obsah pouze prvního modulu CPU.

2.2.2 Společný programový kód aplikace

Základní nastavení proměnných, které se využívají za běhu programového kódu, je uloženo ve skriptu s názvem *„mainvar.sh“*. Zde jsou uvedeny cesty k potřebným programům, k adresářům projektu a dalším nastavením jako například datumy a časová razítka, která budou později použita při ukládání dat do SŘBD. Tento skript je vkládán na vyšších úrovních aplikace tak, aby hodnoty byly dostupné buď v nižších částech programu, nebo se uvedený skript s nastavením vkládá opakovaně, aby bylo dosaženo nové inicializace proměnných, jako např. již zmíněná časová razítka. Níže následuje úryvek kódu uvedeného skriptu.

Aplikační kód 1 - mainvar.sh

```
 1:  hostNameCmd='/bin/hostname'
 2:  dateCmd='/bin/date'
 3:  sqliteCmd='/usr/bin/sqlite3'
 4:  catCmd='/bin/cat'
 5:  awkCmd='/usr/bin/awk'
 6:  psCmd='/bin/ps'
 7:  dfCmd='/bin/df'
 8:  wcCmd='/usr/bin/wc'
 9:  mountCmd='/bin/mount'
10:  scpCmd='/usr/bin/scp'
11:  hostName=$($hostNameCmd)
12:  date=$($dateCmd '+%Y-%b-%d-%H-%M')
13:  curDateTime=$($dateCmd '+%Y-%b-%d %H:%M:%S') # current date and time
14:  tstamp=$($dateCmd +%s) # timestamp
```

```
15:
16:  # monitoring module - project-root/modules/*/
17:  modules=('cpu' 'disk' 'mem' 'net' 'system') # where modules manage-db.sh will
be runing
18:  diskPart=('data' '/' 'var') # name of mountpoints which will be monitored (/,
var, home)...
19:  networkIface=('eth0' 'lo') # interface for seting up DB (eth0, lo, on ESX
host: vmnic0)...
20: ...
```

2.2.3 Moduly zodpovědné za sběr dat

Jedná se o programový kód, který bude zajišťovat přístup k datům buď přímou cestou, tj. načte obsah konkrétních souborů z adresáře /proc, nebo použije výstup z programů speciálně určených pro zobrazení dat z adresáře /proc. Každý modul je navržen tak, aby mohl fungovat i nezávisle na hlavním programovém kódu. S daty získanými každým modulem je uloženo do databáze zároveň datum ve formě časového razítka. Každý modul má vlastní soubor s proměnnými, které mají lokální charakter a jsou platné pouze pro daný modul. Název tohoto skriptu s proměnnými je *„var.sh“*.

2.2.3.1 Modul sledující zátěž procesoru - CPU

Tento modul načítá data přímo ze souboru *„stat“* adresáře /proc do pole hodnot a provede výpočet. Tato akce musí být provedena minimálně 2 krát za sebou, protože se musí vypočítat rozdíly hodnot, ze kterých se určí uvedené využití procesoru. Výsledná zpracovaná hodnota udává procentuální využití procesoru počítače. Kód byl převzat a upraven ze zdroje uvedeného zde [9]. Výkonné jádro skriptu je uvedeno na řádcích 1 - 27 následujícího Aplikační kód 2 - cpu.sh . Programový cyklus se provede 5 krát a až poté je výsledná hodnota uložena do databáze.

Aplikační kód 2 - cpu.sh

```
 1:  while [ $i != 4 ]; do
 2:      CPU=($($catCmd $procItem | $awkCmd '/^cpu / {print}')) # Get the total CPU
statistics.
 3:      unset CPU[0]    # Discard the "cpu" prefix.
 4:      IDLE=${CPU[4]}  # Get the idle CPU time.
 5:
 6:      # Calculate the total CPU time.
 7:      TOTAL=0
 8:      for VALUE in "${CPU[@]}"; do
 9:              let "TOTAL=$TOTAL+$VALUE"
10:      done
11:
12:      # Calculate the CPU usage since we last checked.
13:      let "DIFF_IDLE=$IDLE-$PREV_IDLE"
14:      let "DIFF_TOTAL=$TOTAL-$PREV_TOTAL"
15:      let "DIFF_USAGE=(1000*($DIFF_TOTAL-$DIFF_IDLE)/$DIFF_TOTAL+5)/10"
16:      if [ "$1" != 'q' ]; then
17:              echo -en "\rCPU: $DIFF_USAGE%  \b\b"
18:      fi
19:
20:      # Remember the total and idle CPU times for the next check.
21:      PREV_TOTAL="$TOTAL"
22:      PREV_IDLE="$IDLE"
23:
24:      # Wait before checking again.
25:      sleep 1
26:      i=$(($i + 1))
```

```
27:  done
```

Datový slovník modulu CPU

Tabulka 2

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
cpu	tstamp	Timestamp	-	N	N	-	časové razítko
	value	Integer	-	N	Y	-	naměřená hodnota

2.2.3.2 Modul sledující využití diskových oddílů - DISK

Modul bude pársovat výstup programu DF[4]. To, které oddíly budou aplikací sledovány je nastaveno ve skriptu *mainvar.sh*. Nejsou zde vyjmenovány klasické diskové oddíly, tak jak jsou známy z linuxových systémů, ale názvy přípojných bodů jako např. /var, /home. Na základě tohoto nastavení je vygenerován potřebný počet databázových entit. Modul sleduje následující hodnoty: celkové místo, využití a dostupné místo na disku v bajtech. Výkonné jádro skriptu je vidět na řádcích 2 až 34 Aplikační kód 3 - part.sh. Data jsou napřed přiřazena do proměnné a pak postupně pomocí výrazů jazyka AWK zpracovávána. Kořenový svazek „/“ je překládán na text „root“. Kód skriptu je samopopisný a nepotřebuje dalšího komentáře.

Aplikační kód 3 - part.sh

```
 1:  # gather data from df utility, (local parttions)
 2:  tmpVar=$($dfCmd  -l)
 3:
 4:  for item in ${diskPart[*]} ; do # set in mainvar.sh
 5:
 6:      if [ "$item" == '/' ]; then # translate root part
 7:              tmpItem=''
 8:      else
 9:              tmpItem="$item"
10:      fi
11:
12:      # parse data from df -l output
13:      part=$(printf %b "$tmpVar" | $awkCmd '$6 ~ "'"^/$tmpItem$"'" {print}')
14:      used=$(printf %b "$part" | $awkCmd '{print $3}')
15:      available=$(printf %b "$part" | $awkCmd '{print $4}')
16:      use=$(printf %b "$part" | $awkCmd 'gsub("%", "", $5) {print $5}')
17:
18:      if [ "$item" == '/' ];then
19:              item='root'
20:      fi
21:
22:      sql="insert into $item values($tstamp, $used, $available, $use)"
23:
24:      if [ "$1" != 'q' ]; then
25:              echo
26:              echo "Inserting new value"
27:              echo
28:              echo "$sqliteCmd $dbDir$moduleName$ext $sql"
29:              echo
30:      fi
31:
```

[4] DF zobrazuje detailní využití připojených diskových oddílů.

```
32:      $sqliteCmd $dbDir$moduleName$ext "$sql"
33:
34:  done
```

Datový slovník modulu DISK

Počet entit bude záviset na počtu sledovaných diskových oddílů.

Tabulka 3

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
Název svazku	tstamp	Timestamp	-	N	N	-	časové razítko
	used	Integer	-	N	Y	-	naměřená hodnota
	available	Integer	-	N	Y	-	naměřená hodnota
	use	Integer	-	N	Y	-	naměřená hodnota

2.2.3.3 Modul sledující paměťový subsystém - MEM

Úkolem modulu bude sledování hodnot týkajících se paměťového subsystému počítače, které jsou uloženy v souboru *„meminfo“* adresáře /proc. Obsah souboru je opět načten do proměnné a pomocí programových konstrukcí jazyka AWK pársován na patřičné hodnoty, které jsou po úpravě uloženy do příslušné tabulky databáze. Jsou zde sledovány hodnoty relevantní pro fyzickou paměť systému (RAM), a také pamět odkládacího oddílu (SWAP). Konkrétně jde o hodnoty celkové paměti, použité paměti a paměti dostupné. Získané hodnoty jsou uvedeny v kilobajtech.

Aplikační kód 4 - mem.sh

```
 1:  # gather data from /proc/meminfo
 2:  procItemTemp=$($catCmd $procItem)
 3:  #printf "$procItemTemp"
 4:
 5:  # parse gathered data
 6:  memTotal=$(printf "$procItemTemp" | $awkCmd '$1 == "MemTotal:" {print $2}')
 7:  memFree=$(printf "$procItemTemp" | $awkCmd '$1 == "MemFree:" {print $2}')
 8:  memUsed=$(($memTotal - $memFree))
 9:
10:  swapTotal=$(printf "$procItemTemp" | $awkCmd '$1 == "SwapTotal:" {print $2}')
11:  swapFree=$(printf "$procItemTemp" | $awkCmd '$1 == "SwapFree:" {print $2}')
12:  swapUsed=$(($swapTotal - $swapFree))
```

Datový slovník modulu MEM

Tabulka 4

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
totalmem	tstamp	Timestamp	-	N	N	-	časové razítko
	memtotal	Integer	-	N	Y	-	naměřená hodnota
	swaptotal	Integer	-	N	Y	-	naměřená hodnota

	use	Integer	-	N	Y	-	naměřená hodnota
mem	tstamp	Timestamp	-	N	N	-	časové razítko
	memfree	Integer	-	N	Y	-	naměřená hodnota
	memused	Integer	-	N	Y	-	naměřená hodnota
swap	tstamp	Timestamp	-	N	N	-	časové razítko
	swapfree	Integer	-	N	Y	-	naměřená hodnota
	swapfree	Integer	-	N	Y	-	naměřená hodnota

2.2.3.4 Modul sledující síťový provoz - NET

Informace o počtu přenesených bajtů síťovým rozhraním bude předmětem zájmu tohoto modulu. V hlavním konfiguračním skriptu *mainvar.sh* jsou definována rozhraní, která budou sledována. Samotná data o provozu síťových rozhraní jsou uložena v podadresáři *net* adresáře /proc, konkrétně jde o soubor *dev*. Jako v předchozích případech, je opět načten obsah výše zmíněného souboru do proměnné, ze které jsou postupně získávány hledané hodnoty za pomoci programových konstrukcí jazyka AWK. Hledaná data uvádějí celkový počet přijatých a odeslaných bajtů v době maření.

Aplikační kód 5 - net.sh

```
 1:  for item in ${networkIface[*]} ; do
 2:
 3:      # gather data from /proc/net/dev
 4:      procItemTemp=$($awkCmd ' $0 ~ "'"$item:"'" {print}' $procItem)
 5:      #$printf "$procItemTemp $face"
 6:
 7:      # parse dev data, match() push values into RSTART and RLENGTH
 8:      procItemTemp=$(printf "$procItemTemp" | $awkCmd 'match($0, "'"$item:"'")
{print substr($0, RSTART +  RLENGTH, length($0))}')
 9:      #printf "$procItemTemp"
10:
11:      rx=$(printf "$procItemTemp" | $awkCmd '{print $1}')
12:      tx=$(printf "$procItemTemp" | $awkCmd '{print $9}')
13:
14:      sql="insert into $item values($tstamp, $rx, $tx)"
15:
16:      if [ "$1" != 'q' ]; then
17:              echo
18:              echo "Inserting new value"
19:              echo
20:              echo "$sqliteCmd $dbDir$moduleName$ext $sql"
21:              echo
22:      fi
23:
24:      $sqliteCmd $dbDir$moduleName$ext "$sql"
25:  done
```

Datový slovník modulu NET

Počet entit bude záviset na počtu sledovaných síťových rozhraní.

Tabulka 5

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
Název síťového rozhraní	tstamp	Timestamp	-	N	N	-	časové razítko
	used	Integer	-	N	Y	-	naměřená hodnota
	rx	Integer	-	N	Y	-	naměřená hodnota
	tx	Integer	-	N	Y	-	naměřená hodnota

2.2.3.5 Modul sledující stav systému - SYSTEM

V tomto modulu se nalézají celkem tři skripty, které sledují následující hodnoty:

1. Počet procesů běžících v systému *„num-of-run-proc.sh“*
2. Zatížení v systému *„sysload.sh“*
3. Doba běhu systému *„uptáme.sh“*

První skript spustí unixový příkaz PS, který zobrazí seznam procesů běžících v systému a přes přesměrování vstupu a výstupu je tento mezivýsledek předán programu pro počítání řádků. Od posledního výsledku se musí odečíst ještě procesy, které představují nutnou režii spuštěných programů.

Aplikační kód 6 - num-of-run-proc.sh

```
1:  # gather data from ps utility
2:  numRunProc=$($psCmd ax | $wcCmd -l)
3:  numRunProc=$(($numRunProc - 4)) # - 4 remove self process from count
4:
5:  sql="insert into numrunproc values($tstamp, $numRunProc)"
```

Druhý skript načte obsah souboru *„loadavg“* z adresáře /proc. Tento skript je zajímavý svým minimalistickým využitím systémových příkazů, což vede k efektivnímu běhu a minimální zátěži systému, a tím se příznivě projeví na rychlosti provedení tohoto skriptu. Na řádce č. 1 je vidět méně efektivní způsob přístupu k zpracování dat, kdy pomocí příkazu CAT je obsah souboru přesměrován do „roury“, která jej přesměrovává na vstup interpretu jazyka AWK. Takhle jsou neefektivně zaměstnány tři aplikace (BASH, CAT a AWK). Řádka č. 2 načte obsah souboru do proměnné a na řádce č. 3 je proveden regulární výraz skriptovacího jazyka BASH, který vrátí očekávaný výsledek. Druhá varianta si vystačila jen s příkazem CAT a vlastní instancí příkazového interpretu BASH. Výsledek je opět zaznamenán do databáze. Naměřené hodnoty běhu jednotlivých variant Aplikační kód 7 - sysload.sh jsou k vidění zde [4].

Aplikační kód 7 - sysload.sh

```
1:  #sysLoad=`$catCmd $procItem | $awkCmd '{print $1}'`
2:  sysLoad=$($catCmd $procItem)
3:  sysLoad=${sysLoad%% *}
```

Poslední zjišťovaný údaj není údajem, který by říkal, jak je systém zatížen, ale jen informativně má říci, jak dlouho je systém v chodu od posledního spuštění. O získání této infomace se stará třetí skript s názvem *„uptime.sh“*. Výkonné jádro skriptu je založeno na skriptu publikovaném zde [4]. Do proměnné je načten obsah souboru *„uptime"* z adresáře /proc. Regulární výraz vrátí první sloupec (řádka 2) načtených dat, který reprezentuje počet sekund od spuštění systému. Pak již následuje výpočet (řádky 3 - 6), ve kterém se provede převod na sekundy, minuty, hodiny a dny.

Aplikační kód 8 - uptime.sh

```
 1:  upSeconds=$($catCmd $procItem)
 2:  upSeconds=${upSeconds%%.*}
 3:  let secs=$((${upSeconds}%60))
 4:  let mins=$((${upSeconds}/60%60))
 5:  let hours=$((${upSeconds}/3600%24))
 6:  let days=$((${upSeconds}/86400))
 7:
 8:  if [ "${days}" -ne "0" ]; then
 9:     uptime="${days}d "
10:  fi
11:  uptime="$uptime${hours}h ${mins}m"
```

Datový slovník modulu SYSTEM

Tabulka 6

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
numrunproc, sysload, uptime	tstamp	Timestamp	-	N	N	-	časové razítko
	value	Integer	-	N	Y	-	naměřená hodnota

2.2.4 Automatický sběr dat a údržba databázových souborů

Pro automatizované spouštění opakujících se úloh lze využít softwarový démon CRON, který je implicitně přítomen na každém unixovém systému anebo lze napsat skript, který bude provádět to samé jako CRON, ale ve vlastní režii. Rozhodl jsem se pro druhou variantu u těch úkolů, které tvoří jádro systému aplikace. Jedná se o spuštění skriptů sbírajících data, skriptu, který bude provádět údržbu databázových souborů a skriptu, který provede potřebný převod dat z databázových modulů do databáze, která bude použita pro generování RSS dokumentů.

Výkonným jádrem je skript, který je schopen odpojit své vstupy od terminálu, ve kterém byl spuštěn a provede přesměrování svých výstupů na předem určená místa, konkrétně se bude jednat o soubory, do kterých budou směrovány ladící výstupy aplikace a zprávy o stavu aplikace. Po provedení těchto úkonů skript automaticky přechází do stavu „běh na pozadí“. Takovýto skript lze nazvat *démonovým* procesem.

Démonový skript, jak lze nazvat výše popsaný skript, přijme od uživatele z příkazové řádky argumenty, na základě kterých provede patřičnou programovou konstrukci. Zejména se jedná o tyto klasické argumenty: start, stop.

Kostra Aplikační kód 9 - sysinfo2rss-deamon.sh byla převzata ze zdroje [10].

Aplikační kód 9 - sysinfo2rss-deamon.sh

```
 1:  # daemonize this script
 2:  function daemonize() {
 3:      echo $pid > $tmpDir/$pidFile
 4:      exec 3>&-        # close stdin
 5:      exec 2>>$dbgFile # redirect stderr
 6:      exec 1>>$dbgFile # redirect stdout
 7:      printf "Daemonize $0 with pid: $$ at: $curDateTime \n" >> $dbgFile
 8:  }
 9:
10:  # run module for gathering data, run it every default(5min) period, after 12
iteration run another module
11:  function runModules {
12:      y=0
13:      while [ true ]; do
14:              cd $curDir
15:
16:              . ./run-modules.sh $y # run every $dumInterval interval - default
5min
17:
18:              if [ $y -gt 11 ]; then # run every 60min its depend on default
interval value
19:                      cd $curDir
20:                      . ./dump-mudules-data-to-db.sh q
21:                      . ./maintan-db.sh q
22:                      y=0
23:              fi
24:              y=$(($y+1))
25:              sleep $dumpInterval
26:      done
27:  }
28:
29:  case "$1" in
30:      'start')
31:              if [ ! -f $tmpDir/$pidFile ]; then
32:                      $0 run &
33:                      printf "\n\n Daemon started ... ${GREEN}[ok]${END}\n\n"
34:                      exec 3>&- # close stdin
35:                      exec 2>&- # close stderr
36:                      exec 1>&- # close stdout
37:                      exit 0
38:              else
39:                      $0 status
40:              fi
41:      ;;
42:
43:      'run')
44:              daemonize
45:              runModules
46:
47:      ;;
48:
49:      'stop')
50:              if [ -f $tmpDir/$pidFile ]; then
```

```
51:                         pid=$($catCmd $tmpDir/$pidFile)
52:                         printf "\n\n Process pid: ${GREEN}$pid${END} is stoping
now!\n\n"
53:                         kill $pid
54:                         rm $tmpDir/$pidFile
55:                         printf "\n Process was stoped \n\n"
56:                         printf "Daemon $0 with pid: $pid was stopt at:
$curDateTime \n" >> $dbgFile
57:                 else
58:                         printf "\n\n $0 isn't currently running\n\n"
59:                 fi
60:
61:      ;;
```

Na řádce č. 32 je volána volba „run“, která zajistí přesun běhu skriptu na pozadí a dále jsou uzavřeny standardní vstup, chybový výstup a standardní výstup ř. 34 - 36.

Hlavní programový kód je tvořen nekonečnou smyčkou, která je umístěna ve funkci *„runModules“* (ř. 11 - 27). Ve smyčce jsou spouštěny po sobě moduly načítající data (ř. 16), tato činnost je přerušena na dalších 5 minut příkazem sleep (ř. 25). Tento proces se opakuje 12 krát, což představuje dobu, za kterou se spustí skript (ř. 21), který zajistí na základě definovaných pravidel smazání starých záznamů ze všech databázových souborů aplikace. Další skript (ř. 20), který se spouští po 12 iteracích je skript zajišťující výpočet nad daty v databázových modulech, získaná data naformátuje a uloží do hlavní databáze aplikace.

Základním časovým algoritmem je 5 minutový odstup při měření dat pro moduly a také hodinový, který slouží pro údržbu a provedení výpočetních úkonů spolu se sestavením dat.

Ve většině databázových tabulek náležejících ke konkrétním modulům je po provedení kroků popsaných výše uloženo 12 hodnot, ze kterých se provede pomocí klasických agregačních funkcí jazyka SQL průměr z těchto naměřených hodnot.

U některých hodnot, jako například u statistiky počtu přenesených bajtů, se neprovádí výpočet průměru, ale provede se pouze nominální rozdíl dvou hodnot z 12 naměřených a to maximální a minimální (opět pomocí funkcí jazyka SQL), což vede na počet přenesených bajtů za hodinu.

Vypočítaná data jsou převedena na příslušné jednotky a správný formát. Poté jako jeden řetězec uložena do hlavní databáze aplikace. Za výše popsané úkony je zodpovědný skript *„dump-mudules-data-to-db.sh“* ř. 20 Aplikační kód 9 - sysinfo2rss-deamon.sh.

Jen pro doplnění uvádím, že časová orientace v záznamech uložených v tabulkách databáze se odehrává na základě pořízených časových razítek, tj. například aktuální časové razítko mínus 3600s dovoluje vybrat záznamy mladší než jedna hodina atd.

2.2.5 Generování RSS dokumentu

Hlavní databázový soubor obsahuje upravená a naformátovaná data s informacemi o systému. Každý řádek tabulky představuje jednu hodinu běhu systému. Data (informace) jsou již nachystána ve tvaru potřebném pro zobrazení v textových výstupech. Jedním z posledních kroků je vygenerování RSS dokumentu, který bude nést monitorované informace.

Skript *„make-rss.sh“*, generující RSS dokument je složen ze tří hlavních částí.

1. Část generuje hlavičku RSS dokumentu

```
 1:  # make head of rss document
 2:  function makeRssHead() {
 3:      buildDate=$($dateCmd -R) # rfc date format
 4:      rss='<?xml version="1.0" encoding="utf-8"?> \n'
 5:      rss="$rss"'<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom"> \n'
 6:      rss="$rss <channel>$eof"
 7:      rss="$rss <title>$hostName system state info channel</title>$eof"
 8:      rss="$rss <link>$url/$rssDocument</link>$eof"
 9:      rss="$rss <description>$hostName system state info
channel</description>$eof"
10:      rss="$rss <category>Server monitoring</category>$eof"
11:      rss="$rss <image>$eof"
12:      rss="$rss <title>$hostName system state info channel</title>$eof"
13:      rss="$rss <link>$url/$rssDocument</link>$eof"
14:      rss="$rss <url>$url/img/$feedIcon</url>$eof"
15:      rss="$rss </image>$eof"
16:      rss="$rss <atom:link href=\"$url/$rssDocument\" rel=\"self\"
type=\"application/rss+xml\" />$eof"
17:      rss="$rss <language>cs</language>$eof"
18:      rss="$rss <pubDate>$buildDate</pubDate>$eof"
19:      rss="$rss <lastBuildDate>$buildDate</lastBuildDate>$eof"
20:      rss="$rss <ttl>60</ttl>$eof"
21:  }
```

Hlavička dokumentu vychází z definice struktury RSS.

2. Část zodpovědná za vytvoření položek „item“ RSS dokumentu - samotné informace.

```
 1:  # make item of rss documents
 2:  function makeRssItem() {
 3:      unset rssItem
 4:      rssItem="<item>$eof"
 5:      rssItem="$rssItem<title>$hostName system status</title>$eof"
 6:      rssItem="$rssItem  <link>$url/sysinfo2rss.php?item=$1</link>$eof"
 7:      pubStampDate=$($sqliteCmd $runDir/$dbDir$mainDb$ext "select tstamp from
$mainDb where idRss = $1")
 8:      tstamp2date $pubStampDate # convert timestamp to RFC date
 9:      rssItem="$rssItem  <pubDate>$pubDate</pubDate>$eof"
10:      res=$($sqliteCmd $runDir/$dbDir$mainDb$ext "select value from $mainDb where
idRss = $1")
11:      if [ "$firstParam" != 'q' ]; then
12:              printf %b "$res $eof"
13:      fi
14:      rssItem="$rssItem  <description><![CDATA[$res]]></description>$eof"
15:      rssItem="$rssItem  <guid isPermaLink=\"false\"
>$url/sysinfo2rss.php?item=$1</guid>$eof"
16:      rssItem="$rssItem</item>$eof"
17:  }
```

3. Část zodpovědná za správné zakončení dokumentu

```
1:  # make foot of rss document
2:  function makeRssFoote() {
3:      rss="</channel> $eof"
```

```
4:       rss="$rss</rss>"
5:  }
```

Generování dokumentu se děje automaticky, ale nikoliv pomocí démon skriptu, který byl popsán dříve v přechozí kapitole. Skript *make-rss.sh* je zadán jako jeden z údajů CRON tabulky, kde je spuštěn každou hodinu. Toto nastavení je provedeno pod účtem uživatele, majícího oprávnění k publikování webových dokumentů. Ke konci zpracování uvedeného skriptu se provede zkopírování RSS dokumentu k jeho následnému publikování.

2.2.6 Publikování RSS dokumentu

Uložený RSS dokument je přesunut do složek webového serveru, odkud si jej mohou stáhnout vizualizéry v podobě RSS čteček. O samotný přesun se stará skript s názvem *„upload-rss-to-web-server.sh“*. Tento skript podle nastavení v hlavním konfiguračním skriptu *mainvar.sh* provede:

- lokální kopírování souboru
- kopírování souboru na vzdálený webový server pomocí programu SCP, tak jak bylo popsáno v kapitole 2.1.7.

2.2.7 Pomocné skripty aplikace

Aplikace obsahuje menší počet „pomocných“ skriptů, které mají za úkol zajistit jednorázovou akci a dále se tyto skripty nepoužívají (nejsou zapotřebí pro další chod aplikace).

2.2.7.1 Inicializace dabázových souborů

Adresář *„scripts“*, který je obsažen v adresáři každého modulu, obsahuje skript s názvem *„manage-db.sh“*, který má za úkol inicializaci databázového souboru (každý modul sbírající data má svůj vlastní databázový soubor) a také struktury tabulek, k čemuž je použito klasického jazyka DML. Každý skriptový soubor může být spuštěn samostatně, anebo pomocí „centrálního“ skriptu *„manage-db.sh“*, který je uložen také v adresáři *scripts*, ale tentokrát v kořenovém adresáři projektu.

2.2.7.2 Kontrola kompatibility systému

Skript *„system-compatibility-check.sh“* provede zevrubnou kontrolu nastavení systému. Zejména se jedná o přítomnost použitých programů, nutných pro chod aplikace, možnost čtení dat z adresáře /proc a kontrola některých vybraných nastavení z hlavního skriptu s proměnnými - *mainvar.sh*. Skript po svém běhu upozorní na shodu či neshodu uživatele a doporučí možné kroky k nápravě. Tento skript si neklade za cíl provést vyčerpávající sadu kontrolních testů.

2.2.8 PHP skripty pro zobrazení detailu článku z RSS dokumentu

Monitorovací systém je vybaven sadou několika webových skriptů, které mají za úkol po nahrání na webový server zobrazit detailní náhled na data RSS dokumentu ve formátu xHTML. Tyto skripty se připojí k hlavnímu databázovému souboru a na základě předaného parametru z uživatelovy akce (kliknutí na odkaz v RSS dokumentu) se načte detail informace.

Tyto skripty nejsou nutnou součástí aplikace, jen zvyšují uživatelský komfort vizualizační vrstvy aplikace a slouží pro další možný směr růstu a rozvoje aplikace.

3 Klientská aplikace pro monitorování serverového operačního systému

Pro čtení RSS dokumentů (definice RSS formátu viz. Příloha A), které budou exportovány na straně serveru, lze na klientské straně použít integrovaný prohlížeč webových stránek, který je schopný textový obsah těchto dokumentů zobrazit. Nicméně formátování RSS dokumentů není vhodně uzpůsobeno pro zobrazení na malém displeji kapesního zařízení v prostředí webového prohlížeče. Tento nedostatek lze odstranit naprogramováním nebo použitím vhodné aplikace, která bude schopna RSS dokumenty obsahující monitorovaná data příslušně naformátovat pro zobrazení na malých zobrazovacích jednotkách a bude je umět správně zobrazit.

3.1 Vizualizér - RSS čtečka

RSS čtečku lze obecně definovat takto: jedná se o samostatnou aplikaci, která zpracovává RSS dokumenty, které jsou podtřídou XML. RSS dokument obsahuje strukturované informace z určité oblasti, která je předmětem zájmu uživatele. Uživatel aplikace si přidá informační „kanál“ (zdroj RSS dokumentu, typicky je takovýto dokument dostupný přes HTTP protokol) do čtečky a ta mu přehledně zobrazí data z informačního kanálu. RSS čtečka může obsluhovat více informačních zdrojů. Aplikace mohou obsahovat další vlastnosti pro usnadnění odběru informačních zpráv [8].

3.2 Návrh řešení čtečky RSS dokumentů - popis problémů

Každá aplikace, která zpracovává data ze vzdáleného zdroje, bude muset řešit několik zásadních oblastí k tomu, aby byla dobře a lehce použitelná. V následujících odstavcích popíši zásadní požadavky na to, co by čtečka RSS dokumentů měla splňovat a jaké vývojové platformy, případně technologie, použít k usnadnění vývoje této aplikace.

3.3 Požadavky na aplikaci

3.3.1 Zobrazení a uživatelské rozhraní - UI (user interface)

Zásadním a limitujícím faktorem bude velikost zobrazení, jelikož se jedná o zařízení typu PDA nebo SmartPhone zařízení, postavené na operačním systému WM, kde nelze předpokládat použití větších zobrazení. Typicky, dnešní PDA na platformě WM disponují zobrazením 240 x 320 zobrazených bodů a barevná hloubka displeje je 65 tisíc barev. Tento trend se velice pomalu posouvá k větším zobrazovacím jednotkám. Aplikace bude muset být schopna zobrazit přehledně větší množství informací tak, aby je uživatel nepřehlédl a měl je viditelně uspořádány. Ovládání by mělo být uzpůsobeno tak, aby se prostředí dalo intuitivně a pohodlně ovládat prsty. UI bude muset co nejvíce těžit z takzvaného „touch friendly“ ovládacího rozhraní.

3.3.2 Stahování dat ze serveru

Aplikace bude muset komunikovat po síti se serverem, kde budou dostupná data ve formě RSS dokumentů. Bude se muset vypořádat s tím, když server není dostupný vlivem výpadku sítě. Aplikace musí mít jistou inteligenci, tak aby síťové výpadky byla schopná správně ošetřit. Uživateli by mělo být hned patrno, že aplikace při přenosu pracuje a že aplikace nepřestala reagovat.

3.3.3 Zpracování dat

Data po úspěšném stažení aplikací jsou v surovém stavu. Nejsou okamžitě použitelná pro zobrazení. Aplikace musí být schopna RSS dokumenty stažené ze serveru pársovat, uložit a následně zobrazit. Samotné pársování je proces, kdy aplikace na základě jistých pravidel umí dokument (v tomto případě

RSS) procházet, rozumí jeho struktuře a umí s ním dále pracovat (s jeho celkem nebo částmi). Pársování dokumentů je výpočetně a mnohdy i paměťově náročná záležitost. Cílová architektura musí umět pracovat se strukturou XML dokumentů.

3.3.4 Uložení dat

Zpracovaná data musí být uložena v paměťové struktuře, kde budou zachována i po vypnutí přístroje nebo po výpadku napájení. K této struktuře musí být snadný přístup a musí zajistit konzistenci uložených dat. Data se musí dát snadno modifikovat. Datová struktura by měla být lehce dostupná na cílovém zařízení.

3.3.5 Vyhledávání klíčových slov ve stažených datech

Nedílnou součástí aplikace pro monitorování by měla být možnost jistým způsobem reagovat na stažená data a dát o tom nejlépe vizuelně na vědomí uživateli. Uživatel aplikace bude v rámci aplikace definovat pravidla, na základě kterých dojde k vytvoření vizuální informace a to po zpracování dat.

3.4 Výběr vhodného řešení a vývojové platformy pro tvorbu aplikace

Výše zmíněné klíčové vlastnosti aplikace, tak jak byly definovány výše v textu, plus atributy operačního sytému Windows Mobile, budou do jisté míry ovlivňovat výběr prostředí, ve kterém aplikace bude vytvořena.

Pro snadný vývoj aplikací na platformě operačních systémů WM, jak jsem zmínil v úvodu, lze uvažovat o dvou vývojových architekturách.

3.4.1 Java ME (Java Micro Edition)

První zmíněnou platformou pro vývoj bude Java ME [11]. Mobilní platforma Java ME je podporována na straně WM zařízení, po doinstalování virtuálního stroje Java. Nejedná se o nativní technologii softwarového gigantu Microsoft a není tedy implicitně na této architektuře OS dostupná.

Tato platforma podporuje tvorbu uživatelského rozhraní - UI s podporou „touch friendly“ vrstvy. Přístup k datům po síti je zde také dostupný.

Pro zpracování RSS (XML) dokumentů lze využít dvou API, která jsou schopná zpracovat tyto data. Jedná se o takzvané Pull parsery a DOM parsery. Zpracovat XML dokumenty na této platformě není určitě žádný problém.

JVM neumí nativně pracovat se souborovým systémem zařízení. Tato funkcionalita je přidána v pozdějších profilech Java ME platformy a nemusí být podporována na cílovém zařízení. Další problém je v tom, že každé zařízení nazývá paměťové médium jiným způsobem. Toto je vážné omezení v ukládání dat na platformě Java ME. Dá se ovšem obejít pomocí RecordStor, což je interní úložiště pro data, které je v novějších implementacích schopno uložit až 2GB dat, pokud to hardware zařízení samozřejmě dovolí. Záznamy jsou zde reprezentovány jako pole bajtů. Složitější struktury lze ukládat do tohoto úložiště pomocí tzv. techniky persistence dat.

Dalším neopomenutelným faktem je přítomnost několika kvalitních vývojových IDE pro tvorbu aplikací nad platformou Java ME plus možnost spustit vytvořený kód v emulátoru této platformy. Zde přichází v úvahu NetBeans od společnosti Sun nebo výborné IDE Eclipse. Obě vývojová prostředí jsou schopna spustit kód v emulátoru mobilní platformy Java. Vizuální návrh designu aplikace je

dostupný pouze v IDE prostředí NetBeans. Z mých praktických zkušeností je ovšem lepší psát aplikaci ručně a nespoléhat se na ne vždy ideálně vygenerovaný kód pomocí IDE.

Z výše uvedeného je patrné, že jisté aspekty vývoje aplikace na platformě Java ME budou přinejmenším velice problematické. Ať už se jedná o jistá bezpečnostní opatření při přístupu k síti, tak hlavně o dost rozdílnou úroveň implementace profilů CLDC na různých verzích JVM. Je zde veliká nehomogenita v podpoře služeb a nejistá možnost v přístupu na souborový systém.

V době psaní této práce je aktuální verze Java ME Software Development Kit 3.0.

3.4.2 .NET CF (Compact Framework)

Dnes velice populární technologie .NET je široce nasazovaná pro tvorbu aplikací různého druhu především na platformě rodiny operačních systému Windows. Ať již jde o desktopové aplikace nebo aplikace umístěné na webovém serveru až po aplikace, které jsou spustitelné na „chytrých" mobilních zařízeních majících konektivitu k Internetu přes WiFi nebo pomocí datového připojení k GSM síti. Odvozená část architektury .NET se jmenuje .NET Compact Framework (CF) [12]. Jde cca. o 10% původní .NET platformy.

.NET CF platforma je určena pro vývoj aplikací nad mobilní architekturou WM. Na rozdíl od konkurenční Java ME jde o nativní technologii. Vývojová platforma .NET CF je velice dobře podporována na všech WM zařízeních. Netrpí nectnostmi a omezeními architektury Java ME.

Stejně tak, jako Java ME, tak .NET CF podporuje tvorbu dotykového prostředí. Formuláře uživatelského prostředí jsou doslova protkány metodami nejrůznějších událostí reagujících na akce uživatele.

Přístup k síti je zde řešen unifikovanými rozhraními, které nepůsobí při vývoji problém. Horší je to s detekcí připojení, které se musí řešit ručně [13]. Zpracování dat ve formě XML je bezproblémové a pomocí nových technologií dostupných v poslední řadě .NET CF velice efektivní.

Ukládání dat na této platformě lze řešit pomocí různých technik. Naskýtá se možnost použití souborového systému, ke kterému je snadnější přístup. Data v textové podobě mohou být ukládána/načítána z/do XML souboru uloženého v paměti zařízení. A jako poslední možnost je použití plnohodnotné databáze, která je po doinstalování ovladače nativně podporována.

Přístup k souborovému systému je zde řešen nativně. Zjistit přítomnost paměťové karty v zařízení nepůsobí větší problém a funguje na všech zařízeních z rodiny WM bezproblémově.

Velice pochvalně bych se zmínil o vývojovém prostředí VS (Visual Studio 2008), které má připravenou podporu pro spuštění aplikací nad architekturou WM. Zvláště užitečná funkce je možnost aplikaci spustit na reálném zařízení a v něm provádět ladění této aplikace. Výstupem je aplikace, která je hned použitelná a odzkoušená na reálném hardware, což se v případě Java ME nedá bohužel říct.

Na platformě .NET CF lze vybírat ze tří jazyků - C#, C++, Visual Basic. Pro implementaci aplikace dle požadovaných kritérií a po zvážení předností jednotlivých platforem pro vývoj aplikace nad architekturou WM jsem zvolil platformu .NET CF spolu s C# jazykem, která je velice flexibilní a ve

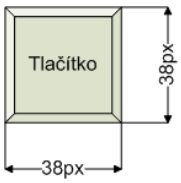
větší vzájemné symbióze s touto mobilní architekturou. Určitě velkým přínosem pro urychlení vývoje je prostředí Visual Studia 2008.

Aktuální verze .NET CF v době psaní této práce je 3.5 SP1.

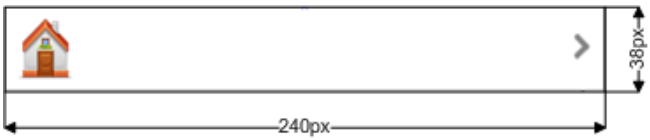
3.5 Návrh aplikace

3.5.1 UI - Uživatelské rozhraní

Uživatelské rozhraní je velice citlivým místem, které zásadně ovlivňuje uživatele a ovládání aplikace. Po spuštění je to hned první věc kterou uživatel vidí a proto návrhu UI musí být věnována maximální pozornost tak, aby výsledné UI bylo pro uživatele co nejvíce lákavé, intuitivní a jednoduché na použití zároveň. Těchto atributů lze dosáhnout pomocí kombinací vhodně zvolených barev, vhodného rozložení ovládacích prvků a vyvážené grafiky jako celku. Ovládací prvky, které se budou ovládat pomocí prstů, musí mít patřičnou velikost. Velikost ovládacích prvků, které se mají snadno ovládat prsty, by mělo být kolem 38 x 38 pixelů. Pro stylus je tento rozměr menší - 21 x 21 pixelů. Návrhy tlačítek zobrazují následující Obrázek 2 a Obrázek 3

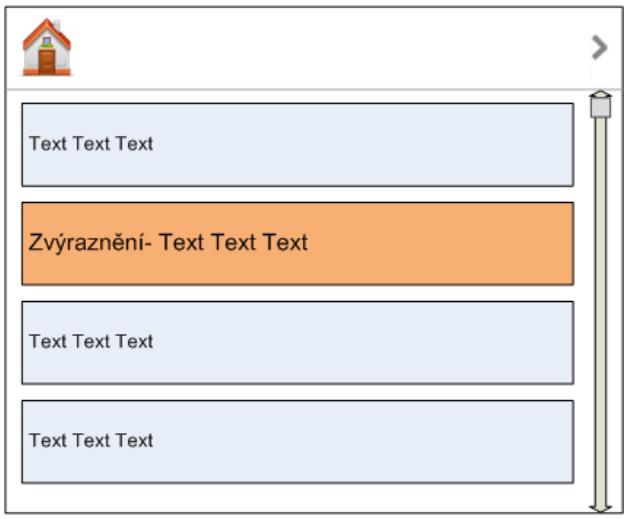


Obrázek 2



Obrázek 3

Vizuální prvky nesoucí informace, které mají na první pohled upoutat, budou patřičně zvýrazněny, tak aby uživatele zaujaly ihned při zobrazení a byly lehce odlišitelné od ostatních informací. Strohá tlačítka, která mohou nést jen textovou informaci, lze nahradit za obrázek s piktogramem doplněným o jednoduchý informativní údaj. Tak se dosáhne poutavějšího grafického podání a zvýší se atraktivita uživatelského prostředí. Návrh zobrazuje Obrázek 4.



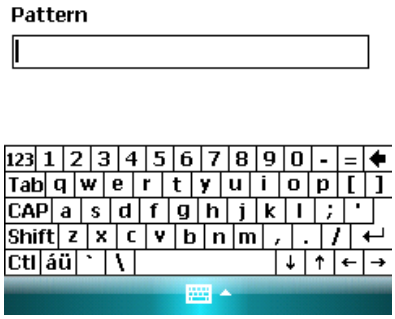
Obrázek 4

V situacích, kdy je aplikace nucena zpracovávat nebo vykreslovat dynamicky informace do prvků umístěných na formuláři aplikace, může dojít k zdánlivému zatuhnutí aplikace. Typicky se tak stane při stahování dat ze sítě, kdy aplikace čeká na data ze serveru atd[5]. Uživatel v takovéto chvíli bude přiměřeně upozorněn (vizuálně) na takovouto aktivitu. Například pomocí blikajícího nápisu, progres baru, nebo kroužícího kolečka, které se animuje tak, aby bylo na první pohled patrné, že aplikace nepřestala pracovat.

Aplikace musí zajistit mnoho činností, a proto se nebude skládat pouze z jediného formuláře UI. Aplikace bude uživateli zobrazovat informaci o tom, kde se právě nalézá, kterou část aplikace teď momentálně využívá. Formuláře budou členěny do logické hierarchie tak, aby byl patrný kontext, ve kterém se uživatel nalézá a byla patrná souslednost z kontextu mezi formuláři. Toho lze na platformě .NET CF dosáhnout nastavením popisku záhlaví formuláře a správným navázáním posloupnosti kroku při přechodu mezi formuláři. Každý formulář bude jednoduše identifikovatelný jednoznačným názvem.

Formuláře, které budou obsahovat ovládací prvky typu textových polí, zobrazí automaticky při umístění kurzoru SIP (Software Input Panel). Tak může být například dosaženo většího komfortu při ovládání aplikace. Výsledek je patrný z následujícího Obrázek 5

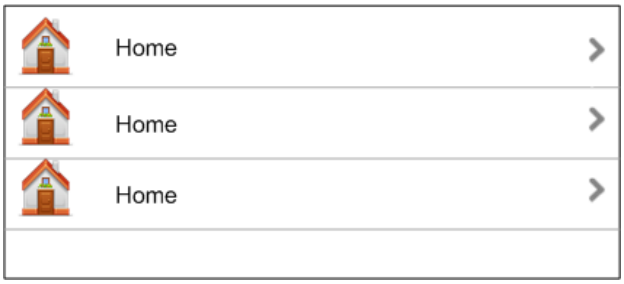
[5] Lze částečně řešit pomocí vláken.



Obrázek 5

Hlavní menu aplikace - *„MainScreen“*, ze kterého se budou aktivovat základní funkce aplikace, může vypadat například následujícími způsoby: Obrázek 6 a Obrázek 7

Obrázek 6



Obrázek 7

Jde o nejpoužívanější vzhledy ovládání na mobilních architekturach, které se dají dobře uzpůsobit pro ovládání pomocí prstů. Prvně uvedený vzhled je známý např. z prostředí Microsoft Windows a druhé uvedené je známo např. z iPhone OS. Jednoznačně nelze říci, které je lepší a které ne. V ovládání pomocí prstů zde nelze pozorovat podstatnějších rozdílů. Jde tedy spíše o osobní vkus tvůrce aplikace. Osobně si myslím, že druhý vzhled je použitelnější, protože výška tlačítek je postačujících 38px a šířka je 240px, což zaručí, že po celé délce obrazovky může být tlačítko aktivováno. Dále tento vzhled bude korespondovat s dalšími vizuálními prvky zobrazovanými v ostatních místech aplikace. Nevýhodou při použití tohoto vzhledu ovládání je větší plýtvání prostorem. Na zobrazenou pracovní plochu aplikace půjde vykreslit jen omezené množství tlačítek, aniž by se musel zobrazit posuvník stránky. Tento nedostatek lze vyřešit pomocí klasických menu[6], kde budou přístupny méně používané části aplikace.

Design aplikace bude navržen pro přístroje s rozlišením 240 x 320 pixelů.

[6] Jinak také zvané - Technické menu

3.5.2 Stažení dat přes síť

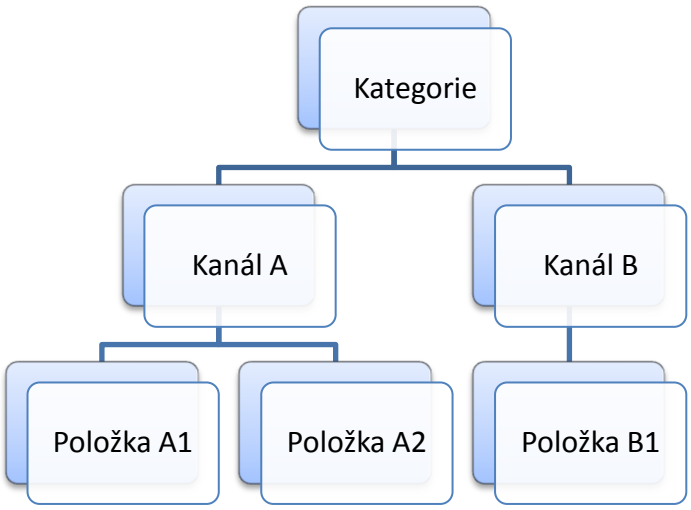
Stahování dat bude zajištěno pomocí samostatných vláken v aplikaci tak, aby bylo možno aplikaci dále používat při stahování dat. Jak bylo zmíněno v kapitole 3.5.1, aplikace vizuálně upozorní uživatele na průběh stahování, tak aby měl jistotu, že aplikace nepřestala pracovat.

Jak bylo naznačeno v požadavcích kladených na aplikaci, aplikace musí umět adekvátně zareagovat na výpadek nebo nepřítomnost sítě či serveru. Aplikace při nedostupnosti sítě zobrazí uživateli přehlednou informaci o tom, že není připojen k síti. Nejlépe pomocí samostatného formuláře s konkrétním popisem chyby.

3.5.3 Zpracování stažených dat

Po úspěšném stažení RSS dokumentu ze serveru bude zajištěno aplikační logikou zpracování dokumentů. RSS dokument bude „dotazován“ (zpracováván) pomocí dotazovacího jazyka LINQ (Language Integrated Query). LINQ je nový dotazovací jazyk na platformě .NET od verze 3.0, který slouží k dotazování nad jakýmikoliv daty, k propojování dat, třídění dat atd. LINQ jazyk pomůže efektivně třídit a adresovat určité části RSS dokumentu. Na základě vytvořených dotazů pomocí LINQ lze ze získaných informací vytvářet patřičné objekty a plnit je pořízenými informacemi. Objekty, které vzniknou výše popsaným postupem, budou v aplikaci reprezentovat části načtených informací RSS dokumentu. Jde o přímočarý přístup k datům, který je vysoce flexibilní, efektivní a výkonný.

Na tomto místě bych zmínil logický model, který bude reprezentovat použitá data v aplikaci. Základem strukturované hierarchie bude Kategorie. Kategorie je objekt pro související objekty typu Kanál. Objekt Kategorie bude udržovat reference na objekty představující jednotlivé kanály přidružené k dané kategorii. Jeden objekt typu Kanál bude obsahovat reference na objekty, které jsou vytvořeny ze staženého RSS dokumentu. Koncové objekty jsou ona zpracovaná a vytěžená data pomocí LINQ. Tato logika bude zřejmá ze struktury RSS dokumentu, která je uvedena v Příloha A. Obrázek 8 zobrazuje hierarchii objektů tak, jak budou reprezentovány v aplikaci. Po zpracování budou data uložena pro pozdější znovuzobrazení.



Obrázek 8

3.5.4 Uložení dat

Aplikační logika zajistí, že vytvořené a naplněné objekty budou ukládány pro pozdější zobrazení, modifikaci či přidáváni dat. Pořízená data budou dostupná při opětovném zapnutí aplikace tak, aby uživatel mohl zpětně procházet již pořízená data. Uložení dat bude další kritickou částí aplikace. Aplikace může například pro textová data používat XML soubor. Zde ale hrozí nebezpečí, že při větším množství dat uložených v XML dokumentu bude docházet k neúměrnému vytěžování procesoru a vysokým paměťovým nárokům na zařízení při jeho zpracování. Aplikace bude mimo textových dat ukládat i data obrazová. Na základě tohoto pohledu se nejeví ukládání obrazových dat přímo do souborového systému jako perspektivní. Výše zmíněné nedostatky lze vyřešit použitím SŘBD.

Pro architekturu WM existuje nativní SŘBD, která se jmenuje SQL CE (Microsoft SQL Server Compact, SQL Server Mobile Edition) [14] a [15]. Jde o volně dostupnou relační databázi, určenou pro desktopové prostředí a pro architekturu WM. Podpora pro SQL CE není implicitně dostupná na WM, ale lze ji jednoduše přidat do zařízení jednoduchou instalací. Podpora pro SQL CE může být také zahrnuta přímo do instalace výsledné aplikace. SQL CE zajistí bezproblémovou datovou vrstvu pro uložení a persistenci dat na platformě WM a to bez zásadnějšího degradování výkonu při větším počtu uložených dat.

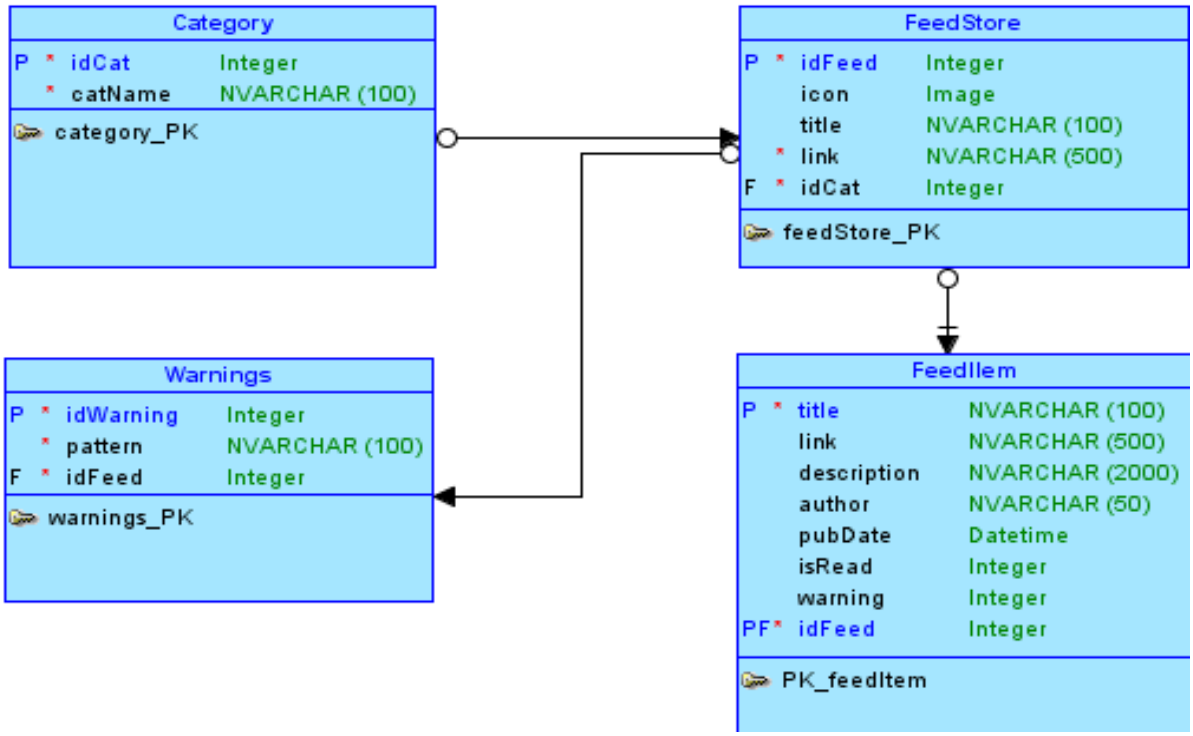
SQL CE je schopen ukládat klasické datové typy – Integer, Varchar, Date, Time a také Blob či Image, atd. Paleta podporovaných typů je velice široká. Samotná databáze je reprezentována jediným souborem s příponou .sdf. Velikost souboru může vzrůstat až do celkové velikosti 4GB. Z prostředí VS 2008 lze velice efektně databázi spravovat. Programátor může vytvářet tabulky, definovat atributy spolu s datovými typy, použít indexy tam, kde je to vhodn, a plánovat další integritní omezení. Provádění dotazů nad vytvořenými schématy z prostředí VS 2008 je podporováno také. Při prvním spuštění aplikace v emulátoru z VS 2008, kde je zapnuta podpora pro SQL CE, je automaticky zajištěno nainstalování podpory pro SQL CE do cílového prostředí.

V kapitole 3.5.3, jsem nastínil hierarchickou strukturu objektů, které nesou informace jak o sobě samém, tak i o dalších objektech či datech. Tyto informace musí být vhodně uchovány a k tomu to účelu bude použit s výhodou SQL CE.

Mimo objekty reprezentující Kategorii, Kanál a položku Kanálu bude existovat objekt udržující klíčová slova pro vyhledání v textu, která budou přidružena ke konkrétnímu Kanálu.

Nyní provedu návrh struktury databáze spolu s integritním omezením. Konceptuální schéma nejlépe vystihne strukturu databáze a případná integritní omezení budou doplněna v datovém slovníku. Návrh konceptuálního schématu pro navrhovanou aplikaci je zobrazen na Obrázek 9

V době psaní této práce je k dispozici SQL CE ve verzi 3.5 SP1.



Obrázek 9

Datový slovník:

Tabulka 7

Entita	atribut	datový typ	délka	KEY	NULL	IO	význam
category	catId	Integer	4	A	N		
	catName	Nvarchar	100	N	N		
feedStore	feedId	Integer	4	A	N		
	icon	Image	-	N	A		
	title	Nvarchar	100	N	A		
	link	Nvarchar	500	A	N		
	catId	Integer	4	N	N		cizí klíč z tabulky category
feedItem	title	Nvarchar	100	A	N		
	link	Nvarchar	500	N	A		
	description	Nvarchar	50	N	A		
	author	Datetime	-	N	A		
	pubDate	Date	7	N	N		
	isRead	Integer	4	N	N	*1)	true/false
	warning	Integer	4	N	N	*1)	true/false
	feedId	Integer	4	A	N		cizí klíč z tabulky feedStore
Warnings	warningId	Integer	4	A	N		
	pattern	Nvarchar	100	N	N		klíčové slovo
	feedId	Integer	4	N	N		

*1) Jde o náhradu booleovského datového typu. Hodnoty budou nabývány z dvouprvkové množiny {0,1}.

3.6 Implementace čtečky RSS dokumentů

V následujícím textu popíši vývoj zásadnějších bloků aplikace dle návrhu a specifikace z dřívějších kapitol a funkcionalitu jednotlivých řešení. Výsledkem bude samospustitelná aplikace, která bude schopna zobrazit data z monitorovaného serveru na architektuře WM.

Budovanou aplikaci lze rozčlenit na jednotlivé stavební bloky, které budou plnit svou vlastní úlohu ve vytvářené aplikaci. Z hlediska UI lze snadno v aplikaci identifikovat potřebný počet formulářů, které je nutno použít jak pro interakci s uživatelem pro vstup informací, tak pro jejich zobrazení.

3.6.1 Formuláře UI

1. Základní obrazovka aplikace (MainForm)
2. Přidání kanálu/kategorie (AddFeedForm)
3. Odebrání kanálů/kategorie (DeleteFeedForm)
4. Aktualizace kanálů (UpdateInfoForm)
5. Zobrazení kanálů (FeedsForm)
6. Zobrazení kanálu (FeedForm)
7. Detail zprávy (FeedItemForm)
8. Přidání hledaného výrazu ke kanálu (AddWarningsForm)
9. Odebrání hledaného výrazu z kanálu (DelWarningsForm)
10. Informační formulář o stavu aplikace – zobrazení při oznámení, chybě atd. (InfoForm)
11. O aplikaci (AboutForm)

Návaznost formulářů UI bude těsněji spjata s dalšími vrstvami aplikace, jako například s datovou vrstvou, vrstvou zodpovědnou za stahování dat a s dalšíma pomocnýma funkcema aplikace, které budou s formuláři UI spolupracovat.

Každý formulář obsahuje vlastní aplikační logiku, která je poplatná funkci formuláře.

Název projektu jsem vybral jako složeninu klíčových slov: Sys, Info, RSS, Reader - *„SysInforRssReader“*.

VS 2008 automaticky vygeneruje třídu „*Program*“ po založení projektu, která je vstupním bodem aplikace. Tato třída obsahuje statickou metodu „*Main*“, která je hledána při startu běhovým prostředím - CLR. Část třídy je zobrazena v následujícím segmentu kódu.

Aplikační kód 10 - Program.cs

```
 1:  namespace SysInfoRssReader
 2:  {
 3:      public class Program
 4:      {
 5:          private MainForm _mainForm;
 6:          public AboutForm AboutForm;
 7:          public AddFeedForm AddFeedForm;
 8:          public DeleteFeedForm DeleteFeedForm;
 9:          public FeedsForm FeedsForm;
16:          ...
```

```
17:          public Program()
18:          {
19:              _mainForm = new MainForm(this);
20:          }
21:
22:          [MTAThread]
23:          static void Main()
24:          {
25:              Program pr = new Program();
26:              Application.Run(pr._mainForm);
27:          }
28:
29:          public MainForm GetMainForm
30:          {
31:              get { return _mainForm; }
32:          }
33:      }
34:  }
```

Z úryvku kódu třídy Aplikační kód 10 - Program.cs je patrno, že jsou zde definovány potřebné formuláře jako proměnné se samopopisným názvem. Z názvu proměnné by mělo být hned patrno k čemu je formulář určen a jaká bude jeho úloha v aplikaci. V konstruktoru třídy je vytvořena instance třídy hlavního formuláře „*MainForm*“, které je předána instance třídy *Program* a to proto, aby bylo možno přistupovat ke všem objektům v aplikaci vytvořeným. Skrze použití tohoto hierarchického objektového modelu lze jednoduše používat reference na objekty v kterémkoliv místě aplikace. Pro úplný přístup k proměnné *MainForm* je vytvořen getter, který zaručí její dostupnost v dalších částech kódu.

3.6.2 Tvorba UI

Pro tvorbu formulářů může být použito dvou odlišných postupů:

1. Design formuláře pomocí WYSIWYG módu VS 2008
2. Ručním psaním kódu formuláře

První způsob je vhodný pro statické formuláře bez požadavků na generování dalších ovládacích prvků dynamicky. Druhý způsob se uplatní u formulářů, u kterých dochází k dynamické tvorbě prvků formuláře, které jsou generovány na základě nějaké interakce s uživatelem nebo v případě, kdy vizuální návrh selhává nebo je nedostatečný pro konkrétní situaci návrhu formuláře.

Každý formulář je v podstatě klasická třída, která se skládá ze tří samostatných souborů začínajících názvem formuláře:

1. Soubor s aplikační logikou (*.cs)
2. Soubor s inicializací ovládacích prvků formuláře jen v případě, kdy je použit WISIWYG mód VS, v opačném případě je zde jen základní nastavení formuláře (*.Designer.cs)
3. Soubor, který může obsahovat metainformace přidružené k formuláři, jako například textové řetězce, obrázky, atd. (*.resx)

Soubor s metainformacemi (zdroji) je vhodný pro uložení statických obrázků, jako například obrázky ovládacích prvků UI. Pokud by aplikace používala internacionalizaci formulářů, tak by zde byli uložené jazykové mutace řetězců pro jednotlivá národní prostředí.

Mimo doménově lokální soubory „zdrojů“ je k dispozici doménově „globální“ soubor, ke kterému je možno přistupovat odkudkoliv ze zdrojového kódu. To skýtá velikou výhodu pro objekty s celoaplikačním významem, tak aby se například opakující se obrázky nemusely vkládat do vícero souborů „zdrojů“ v aplikaci. Tímto způsobem je zajištěno šetření s prostorem a konečným efektem je menší velikost aplikace. Následující úryvek kódu naznačuje použití této funkcionality uvnitř aplikace.

Aplikační kód 11

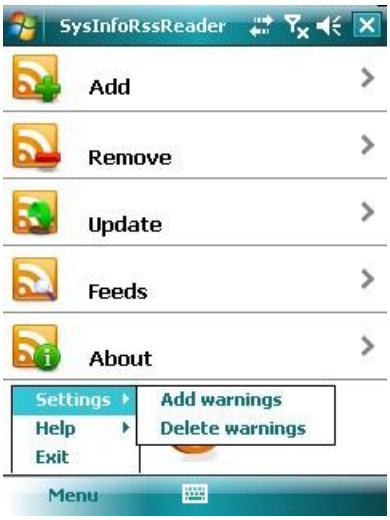
```
 1:  private readonly System.ComponentModel.ComponentResourceManager _resources;
 2:
 3:  ...
 4:          public MainForm(Program program)
 5:          {
 6:              _resources = new
System.ComponentModel.ComponentResourceManager(typeof(Resources));
 7:          }
 8:  ...
 9:
10:          private void DrawHomeBtn(int startPoint)
11:          {
12:              _pictureBox = new PictureBox();
13:              _pictureBox.Image = ((Image)(_resources.GetObject("GoHomeBtn")));
14:  ...
15:          }
16:      }
17:  }
```

3.6.3 Hlavní menu aplikace – MainForm

MainForm tvoří základní bod uživatelského prostředí, které se vykreslí na obrazovku zařízení a s kterým uživatel pracuje, dále jsou zde vytvořeny instance ostatních formulářů, jakož i instance dalších subsystémů aplikace. Design MainForm třídy byl vytvořen pomocí WISIWYG módu VS 2008. Inicializace a konfigurace ovládacích prvků je tady umístěna v samostatném souboru *MainForm.Designer.cs*. Předlohu hlavního menu se stal Obrázek 7 z kapitoly 3.5.1.

U každého tlačítka je k události „Click“ zaregistrován příslušný delegát svázaný s metodou, která je vyvolána jako reakce na událost kliknutí. Tento princip ovládání platí všeobecně pro všechny prvky a napříč všemi formuláři, od kterých se očekává interakce s uživatelem a obsahují-li událost typu „Click“.

Součástí MainForm je Technické menu, které je umístěno v levé spodní části zobrazení, tak jak je známo z rozhraní WM. Výsledné UI MainForm formuláře zobrazuje Obrázek 10.



Obrázek 10

Do Technického menu byly začleněny formuláře a události, které nejsou tak často používány. Zde je patrné slabé místo architektury WM, kde položky Technického menu jsou drobné a hůře se ovládají pomocí prstů. Tohle je neblahý rys celé architektury WM, který bude odstraněn v následujících vydáních mobilní architektury od společnosti Microsoft.

3.6.4 Přidání kanálu RSS - AddFeedForm

Rozvržení formuláře je vytvořeno pomocí WYSIWYG módu VS. AddFeedForm je volán z MainForm (Příloha C, třída MainForm, řádky 155-163), kde je zároveň inicializován. Úlohou tohoto formuláře je, jak vyplývá z názvu, přidání informací o vzdáleném RSS dokumentu. Klíčovým údajem, který se zde zadává, je URL adresa. Je to jediný povinný údaj na formuláři. Jeho nevyplnění znamená nemožnost přidání vzdáleného zdroje. Další informace, jako název poskytovatele RSS dokumentu (název kanálu), je volitelná. Pokud je vyplněn název kanálu, tak je upřednostněn tento zadaný název. Kategorie, do které lze kanál začlenit, je také volitelně vyplnitelnou položkou. Pokud není zadána, je použitá „prázdná kategorie“ bez názvu. V případě, že uživatel nezadá URL zdroje, tak je uživateli zobrazen formulář s varováním k zadání validního zdroje. V případě nezadání názvu kanálu je po kliknutí na tlačítko „Přidat“ zjištěn název kanálu a následně použit jako identifikátor tohoto zdroje. Tlačítko „Přidat“ spustí obslužnou rutinu, která stáhne a rozpársuje RSS dokument a vytvoří základ hierarchie objektu typu kategorie - kanál. Dále uloží objektovou reprezentaci získaných dat do databáze. Všechny tyto moduly a obslužný kód budou popsány dále v následujících statích.

Níže uvedený kód zobrazuje jednu z možností jak řešit automatické zobrazení softwarové klávesnice SIP při umístění kurzoru do textového pole:

Aplikační kód 12

```
1:   private void AddFeedForm_Load(object sender, EventArgs e)
2:           {
3:               tbUrl.GotFocus += TbUrlGotFocus;
4:               tbUrl.LostFocus += TbUrlLostFocus;
5:           }
6:
7:   private void TbUrlLostFocus(object sender, EventArgs e)
8:           {
```

```
 9:              inputPanel1.Enabled = false;
10:          }
11:
12:   private void TbUrlGotFocus(object sender, EventArgs e)
13:          {
14:              inputPanel1.Enabled = true;
15:          }
```

Poté, co uživatel klikne na textové pole, tak zaregistrovaní delegáti (řádek 3 a 4) zajistí aktivaci SIP. Výše popsaný kód je použit všude v aplikaci, kde je přítomno textové pole.

3.6.5 Odebrání kanálu RSS - DeleteFeedForm

Tento formulář je vytvořen pomocí programového kódu a není vygenerován pomocí VS. Na formulář je vykreslen prvek typu panel a na něj se dynamicky přidávají všechny kategorie a v nich obsažené kanály. Tento princip je použit i na dalších místech aplikace a jde o správný modelový návrh skládání ovládacích prvků na formulář. Formulář používá k reprezentaci kategorií a kanálů dva typy prvků - Label a LinkLabel. LinkLabel umožňuje registraci metody k události Click, což prostý prvek Label nedovede. Uživatel vybere kanály, které chce odebrat z aplikace pomocí zatrhávacích polí. Po klepnutí na tlačítko „Odebrat“ jsou kanály odebrány z kategorie. Pokud kategorie je po odebrání kanálů prázdná, tak je kategorie odebrána taktéž. Po provedení akce „Odebrat“ musí být obsah formuláře (panelu) znova načten, aby reflektoval stav hierarchie objektu kategorie/kanál a dále se musí tato změna promítnout do databáze.

Aplikační kód 13 zobrazuje jednoduchý způsob průchodu skrze kolekci zatrhávacích prvků a zjištění jejich stavu (zatrženo/nezatrženo):

Aplikační kód 13

```
1:  private List<CheckBox> _checketFeedItem;
 2:
 3:  public void DrawFeedCheckBox(Feed feedItem, int startPoint)
 4:          {
 5:              _checkBox = new CheckBox();
 6:              _checketFeedItem.Add(_checkBox);
 7:          }
 8:  ...
 9:  foreach (CheckBox item in _checketFeedItem)
10:          {
11:              if(item.Checked)
12:              {
13:                 ...
14:              }
15:          }
```

3.6.6 Aktualizace dat kanálů - UpdateInfoForm

Ještě před tím, než se vykreslí formulář a započne se případně se stahováním dat ze sítě, je provedena detekce dostupnosti připojení k síti. Následující úsek Aplikační kód 14 zobrazuje jednu z možností, jak provést detekci spojení na platformě .NET CF.

Aplikační kód 14

```
 1:  public void GetConnection()
 2:  {
 3:      try
 4:      {
 5:          string hostName = Dns.GetHostName();
 6:          IPHostEntry hostEntry = Dns.GetHostEntry(hostName);
 7:          string hostIpAdd = hostEntry.AddressList[0].ToString();
 8:          _connected = (hostIpAdd != IPAddress.Parse("127.0.0.1").ToString());
 9:      }
10:      catch (Exception e)
11:      {
12:          string x = e.Message;
13:      }
14:  }
```

.NET CF platforma nemá vlastní mechanismy pro detekci přítomnosti sítě. Výše uvedený kód je velice primitivní a zjišťuje pouze přítomnost jiné IP adresy než je adresa lokálního síťového rozhraní v OS. Tato detekce není 100%, protože nedokáže podchytit případ, kdy systému je přidělena nějaká další adresa, a přesto systém nemůže komunikovat s celosvětovou sítí Internet. Další možností, jak napravit tento nedostatek, je možnost provést příkaz ping na vzdálený server nebo překlad IP adresy na jmenný název, ale ani tyto metody nezaručí 100% garanci detekce spojení se sítí a jejich implementace bude o řád složitější.

Formulář UpdateInfoForm má pouze informativní povahu. Je aktivován z hlavního menu aplikace MainForm (Příloha C, třída MainForm, řádky 204 - 205), se kterým dále spolupracuje. V průběhu stahování a zpracovávání dat uživatele vizuálně informuje o probíhající činnosti. Na formulář jsou vykresleny všechny kanály a postupně je každý kanál při činnosti podbarven konkrétní barvou. Samotné stahování dat ze sítě je prováděno na samostatném vlákně (Příloha C, třída MainForm, řádky 206 - 207), aby bylo možno dále s aplikací komunikovat.

.NET CF nedovoluje práci s prvky formulářů mimo hlavní vlákno, ze kterého byl formulář spuštěn. Proto, aby bylo možno manipulovat i s prvky mimo hlavní vlákno, slouží následující úryvek Aplikační kód 15.

Aplikační kód 15

```
 1:  private delegate void UpdateEventDelegate(string msg);
 2:
 3:  public void UpdateEventLabel(string msg)
 4:  {
 5:      if (_labelTemp.InvokeRequired)
 6:         {
 7:            var d = new UpdateEventDelegate(UpdateEventLabel);
 8:            _labelTemp.Invoke(d, msg);
 9:         }
10:         else
11:         {
12:            _labelEvent.Text = msg;
13:            _labelEvent.Refresh();
14:         }
15:  }
```

Jak je patrno z úryvku, zjišťuje se, zdali prvek běží v hlavním vlákně a jestli je třeba provést metodu *Invoke* prvku (řádky 5 - 9).

Poté, co se u nově vytvořeného vlákna zaregistruje metoda „*UpdateFeeds*“, započne samotné zpracovávání dat. Metoda UpdateFeeds (Příloha C, třída MainForm, řádky 294 - 315) prochází postupně kategorie a k nim náležející kanály a provede pro každý objekt kanálu stažení RSS dokumentu. V metodě UpdateFeeds je volána metoda „*DownloadFeedData*“ ze statické třídy „*Downloader*“ (Příloha C, třída Downloader, řádky 16 - 57). Metoda DownloadFeedData využívá běžné *WebRequest* rozhraní z .NET CF. Proto, aby bylo možno komunikovat s webovými servery přes protokol HTTPS, aniž by jejich veřejný klíč byl přítomen v úložišti důvěryhodných serverů v systému, obsahuje aplikace pomocnou třídu *AcceptAllCertificatePolicy* (Příloha C, třída AcceptAllCertificatePolicy). Třída nastaví bezpečnostní politiku, která bude akceptovat i ty certifikáty, které nejsou důvěryhodné. Kód třídy byl publikován zde [16].

Bohužel .NET CF obsahuje jednu nepříjemnou chybu [17], která zapříčiňuje pád aplikace s touto výjimkou: „Nelze číst data z připojení. System.Net.Sockets.SocketException: Neznámá chyba (0x0)“. Tato chyba se vyskytuje ve spojitosti s implementací některých webových serverů, které náhodně pošlou zpět WebRequest rozhraní prázdný zašifrovaný paket. V době psaní této práce není veřejně dostupný HotFix tohoto problému.

Data staženého RSS dokumentu jsou předána metodě, která se postará o jejich další zpracování - pársování. V této chvíli se dostane ke slovu výborný jazyk LINQ, který zajistí rychlé a efektivní vytěžování dat z RSS dokumentu. Ve chvíli, kdy probíhá samotné pársování dat pomocí LINQ, jsou ihned tvořeny „*item*“ objekty, což jsou samotná data kanálu. Tyto objekty jsou následně začleněny k příslušnému kanálu, pro který se právě provádělo pársování dat. Činnost párseru bude patrná z následujícího úryvku Aplikační kód 16.

Aplikační kód 16

```
 1:  public static void ParseFeedItemData(string xmlData, Feed feed, Program
program)
 2:  {
 3:    List<Feed> feeds = null;
 4:
 5:    if (!string.IsNullOrEmpty(xmlData))
 6:    {
 7:        {
 8:            XDocument feedData = XDocument.Parse(xmlData);
 9:
10:            try
11:            {
12:                feeds = (from channel in feedData.Descendants("channel")
13:                         select new Feed
14:                         {
15:                             FeedItems = (from item in
channel.Descendants("item")
16:                                          select new FeedItem
17:                                          {
18:                                              Title =
((string)item.Element("title")).Trim(),
19:                                              Description =
((string)item.Element("description")).Trim(),
```

```
20:                                                 PubDate =
Utils.ParseDateToDbDate((string)item.Element("pubDate")),
21:                                                 Link =
(string)item.Element("link"),
22:                                                 Author =
(string)item.Element("author"),
23:                                                 Warning =
Utils.FindPattern(feed.Warnings, ((string)item.Element("description")).Trim())
24:                                             }).ToList(),
25:                             }).ToList();
26:             }
38:  }
```

Celý kód třídy „*ParseRss*“ je uveden v Příloha C.

O objekty, které byly naplněny daty skrze třídu ParseRss se postará třída „*FeedStoreDB*“, která zajišťuje persistenční vrstvu dat aplikace. Třída FeedStoreDB provádí skrze příslušné metody DML operace nad tabulkami SŘBD. Hlavní doménou této třídy je sestavení objektového modelu reprezentace dat dle kapitol 3.5.3 a 3.5.4 z dat uložených v SŘBD. Třída FeedStoreDB je umístěna v Příloha C. Data ze SŘBD jsou poskládána do již zmíněné objektové hierarchie. Základem je třída *Feeds* (kořenový objekt), která udržuje list objektů třídy *Category* (kategorie) a ty odkazují na objekty typu *Feed* (kanál) a objekty Feed dále odkazují na své objekty typu *FeedItem* (koncová data kanálu). Tento datový model je patrný ze tříd:

- Feeds
- Category
- Feed
- FeedItem

Všechny výše zmíněné třídy jsou součástí Příloha C.

3.6.7 Zobrazení stažených dat - FeedsForm, FeedForm FeedItemForm

Ve fázi, kdy předchozí třídy předzpracovaly potřebná data, přišel čas na jejich zobrazení uživateli. Data jsou uživateli postupně zobrazována ve třech po sobě jdoucích formulářích.

Uživatel z hlavního formuláře MainForm vybere zobrazení kanálů, kde se zobrazí seznam kategorií a kanálů. Uživatel se noří hlouběji v aplikaci po vybrání konkrétního kanálu, až se dostane na výpis titulků z vybraného kanálu. Formulář si po výběru uživatelem uchovává referenci na vybranou položku pro pozdější navazující práci.

Formuláře jsou dynamicky generovány pomocí řízeného kódu aplikace na základě uživatelské interakce. Ovládání je zde zcela podmaněno dotykovému rozhraní, a pouze pro zpětnou kompatibilitu je ve spodní části zobrazení vykresleno Technické menu, které slouží pro přechod zpět na předchozí formulář. Uživatel toto technické menu nemusí ovšem vůbec použít díky tomu, že formuláře mají své logické členění, ve kterém je možno přehledně se vracet zpět pomocí zvětšených ovládacích prvků. Logika generování ovládacích prvků (Panel, LinkLabel, Label, atd.) je obdobná jako ve dříve jmenovaném formuláři DeleteFeedForm. Každý formulář je pouze upraven pro konkrétní účely zobrazení. Jako reprezentativní kód generování formuláře pro zobrazení kanálu jsem vybral *FeedsForm*, který je uveden v Příloha C.

3.6.8 Vyhledání klíčových slov - AddWarningsForm, DelWarningsForm

Uživateli aplikace je nabídnuta možnost zadat ke konkrétnímu kanálu klíčová slova, která jej zajímají a na které si přeje být upozorněn. O částečnou obsluhu této funkcionality se stará formulář AddWarningsForm. Tento formulář byl vytěsněn do Technického menu, protože nebude tak často používán jako významnější položky aplikace. Jak jsem naznačil výše, uživatel skrze tento formulář zadá klíčová slova ke kanálu[7]. Pro výběr položek kanálů jsem zvolil ovládací prvek typu ComboBox. Ve chvíli, kdy jsou aktualizována data aplikace[8], je zároveň prohledáván text „*description*“ položek „*item*“ na výskyt těchto klíčových slov. Níže uvedený kód při výskytu klíčového slova nastaví pravdivostní hodnotu.

Aplikační kód 17

```
 1:  public static int FindPattern(ArrayList patterns, string text)
 2:  {
 3:      int warning = 0;
 4:
 5:      if (!text.Equals(null))
 6:      {
 7:          foreach (var pattern in patterns)
 8:          {
 9:              if (text.Contains(pattern.ToString()))
10:              {
11:                  warning = 1;
12:              }
13:          }
14:      }
15:
16:      return warning;
17:      }
18:  }
```

Hodnota vrácená výše uvedeným kódem je poznačena u položky item. Při zobrazení titulku kanálu, který obsahuje klíčové slovo, je tato položka odlišena od ostatních jinou barvou pozadí právě na základě výše zmíněného příznaku. Jde o efektivní způsob, jak dynamicky odlišit zobrazená data.

Analogicky *DelWarningsForm* slouží pro odebrání klíčových slov z kanálu. Ve chvíli, kdy uživatel odebere poslední klíčové slovo z kanálu, aplikační logika zajistí zákaz ovládání prvku rozevíracího listového seznamu.

3.6.9 Informování uživatele - InfoForm

Ve chvíli, kdy aplikace narazí na nějaký problém, ať už takový, že uživatel nezadal potřebné informace, nebo není detekována síť, je zobrazen informativní formulář. Funkce formuláře je velice jednoduchá a přímočará - podat uživateli zřetelně informaci, kde nastal v aplikaci problém. Tento formulář byl do aplikace začleněn proto, aby informativní hlášení šly snadno ovládat pomocí prstů. Samotné .NET CF obsahuje komponentu, která se nazývá „MessageBox“. Tato komponenta se špatně ovládá prsty, a proto je nahrazena informativním formulářem, který dodržuje touch paradigma. Aplikační logika formuláře je napsána ručně pro umožnění dynamického zobrazení prvků formuláře.

[7] Kanál musí být již v aplikaci zaveden.
[8] Probíhá pársováni.

3.6.10 O aplikaci - AboutForm

Formulář z názvu je jen doplňkem a má pouze informativní charakter. Byl vygenerován pomocí WYSIWYG módu VS.

3.6.11 Nápověda

Nejde o samostatný formulář v pravém slova smyslu. Zde jsem použil funkcionalitu architektury WM, kde soubory nápovědy jsou otevírány v externí aplikaci a měly by být uloženy v systémovém adresáři WM zařízení. Soubor nápovědy je specificky naformátovaný HTML dokument. Následující část Aplikační kód 18 způsobí otevření souboru nápovědy v přidružené aplikaci.

Aplikační kód 18

```
Help.ShowHelp(this, @"\windows\SysInfoRssReaderHelp.htm#obsah");
```

Soubor s nápovědou je součástí Příloha C.

Aplikace obsahuje další třídy a obslužné metody, které již nejsou podstatné z pohledu této části práce, která se zaměřuje na aplikaci pro zobrazení monitorovaných dat ze serveru.

Tlačítka uživatelského prostředí jsem vytvořil pomocí grafického programu. Piktogramy na těchto tlačítkách jsou obrázky z volně dostupných zdrojů přístupných na Internetu [18] a [19], které jsem přizpůsobil potřebám aplikace.

4 Závěr

Díky zvolenému formátu RSS pro publikování dat pořízených uvnitř linuxového systému se podařilo vybudovat menší monitorovací systém, který je schopen zobrazit informace napříč portfóliem moderních operačních systémů, webových služeb až po chytré mobilní zařízení, viz. Příloha B.

4.1 Monitorovací aplikace „sysinfo2rss“

Použití RSS dokumentu k vizualizaci informací o stavu monitorovaného systému je aplikováním moderních trendů ve světě informačních technologií. Jde o neotřelý náhled na nové trendy, které nás neustále obklopují.

Aplikace je napsána pomocí standardních nástrojů, které jsou přítomny na novějších operačních systémech Linux. To umožní širší nasazení napříč linuxovými distribucemi. Uživatel znalý zběžné správy OS bude schopen rychle tento nástroj nasadit a sledovat chování systému. Aplikaci a programové kódy zde uvedené lze také chápat jako kostru, ze které mohou vycházet následná rozšíření této aplikace.

Příkazový interpret BASH je neocenitelný v oblasti systémové správy unixových systémů a mnoho mírně pokročilejších správců systémů si bude schopno pro tuto aplikaci napsat další rozšíření tak, aby více vyhovovala jejich potřebám. Použití interpretu BASH s sebou nese i některá úskalí při psaní aplikace. Jedná se například o to, že jazyk není přísně typový a je procedurálně zaměřen. Jistá vylepšení by byla vhodná na místech, kde se opakují určité úseky kódu, ve kterých by se v budoucnu dala uplatnit vyšší míra abstrakce. Příkladem může být detekce přípojných míst v systému Linux.

Kladným rysem aplikace bezpochyby je, že může běžet s oprávněními běžného uživatele[9].

Budoucí vývoj aplikace může jít cestou napsání nových modulů pro sledování dalších hodnot systému Linux, anebo může být přidán další modul aplikace, který bude zodpovědný za detekci OS a portovat aplikaci na další unixové systémy. Výraznějších vylepšení může doznat mechanismus zodpovědný za plánované časové spouštění modulů a obslužných skriptů. Možné vylepšení aplikace může být formou adoptování formátu ATOM, který by mohla aplikace generovat namísto RSS formátu. Novější formát ATOM umožňuje vkládání dalších dat do dokumentu a tím by se nabízela cesta ke vkládání dalších metainfomací. Dalším směrem vývoje může být těsnější propojení se skriptovacím jazykem PHP, který může zajistit vizualizaci dat formou grafů a případně přidat podporu pro další vizuální vylepšení.

Skript zajišťující běh na pozadí by mohl být v budoucnu poupraven pro automatické spuštění a ukončování při startu systému.

Monitorovací aplikace byla otestována a zprovozněna na následujících systémech, kde slouží pro jejich denní kontrolu:

- feinuke.vsb.cz - distribuce Linux Ubuntu, server slouží jako server pro verzování dokumentů na Fakultě elektrotechniky a informatiky
- burns.vsb.cz a barney.vsb.cz - jedná se o ESX 4.0 virtualizační platformu firmy VMware nasazené na Fakultě elektrotechniky a informatiky

[9] Při splnění několika předpokladů.

- ovak.cz - RedHat enterprice server společnosti OVAK a.s. sloužící jako webové sídlo pro prezentaci této společnosti

Aplikaci „sysinfo2rss“ jsem se rozhodl uvolnit pod licencí GNU GPL a zdrojové kódy jsou dostupné na stránkách aplikace: https://feinuke.vsb.cz/stokpa/projects/bp/sysinfo2rss/

4.2 RSS čtečka - „SysInforRssReader“

Vizualizér RSS dokumentů ve formě RSS čtečky byl vytvořen se záměrem vytvořit prototyp, který bude splňovat kritéria kladená na vizualizaci dat nad mobilní architekturou Windows Mobile. Vytýčených cílů bylo dosaženo pomocí vývojového prostředí Microsoft Visual Studia 2008 a vývojového frameworku .NET CF. Aplikace respektuje veškeré programovací doktríny komponentově orientovaného světa.

Pokročilé použití SQL CE jako perzistenční vrstvy aplikace se ukázalo jako vhodná volba s ohledem na hromadné zpracování dat, které nemusí být striktně jen textového charakteru. Data jsou uložena jen v jednom souboru a aplikace tak není nucena z trvalé paměti přístroje načítat desítky informací, které by byly rozprostřeny do menších souborů.

Při vývoji uživatelského rozhraní se podařilo z 90 procent zachovat ovládání pomocí prstů, což je kladným rysem aplikace. Uživatel není nucen používat stylus pro ovládání aplikace.

RSS čtečka je plně funkční a plní očekávanou funkci, pro kterou byla zkonstruována. Možný vývoj této aplikace se může ubírat směrem přidávání dalších funkcionalit jako například automatické stahování dat z přednastavených informačních kanálů. Zobrazení obrázků, které mohou být k textovým datům přidány. Určitou fází pročištění by měl projít kód aplikace tak, aby bylo dosaženo efektivnějšího chodu aplikace a dosáhlo se větší stability.

Aplikace by mohla být v budoucnu obohacena o další formáty obdobného charakteru jako je například novější formát ATOM.

Aplikace by mohla umožňovat detekci dostupnosti nové aktuálnější verze po síti a upozornit uživatele na tuto skutečnost.

Ke změně by mohlo dojít v části zodpovědné za vizuální upozornění uživatele na zaneprázdnění aplikace. Aktuální řešení by mohlo být nahrazeno za známý panel „načítání“, tak jak jej známe ze stolních operačních systémů.

5 Literatura

[1] Nagios Overview. *Nagios.* [Online] 7. Duben 2010. [Citace: 7. Duben 2010.] http://www.nagios.org/.

[2] About. *Zabbix.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.zabbix.com/documentation/1.8/manual/about.

[3] What is MRTG. *MRTG.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://oss.oetiker.ch/mrtg/doc/mrtg.en.html.

[4] **Orr, Giles.** Bash Prompt HOWTO. [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.faqs.org/docs/Linux-HOWTO/Bash-Prompt-HOWTO.html#AEN728.

[5] **Helmut, Herold.** *awk & sed Příručka pro dávkové zpracování textu.* s.l. : Computer Press, 2004. 80-251-0309-9.

[6] SQLite. *Wikipedie, otevřená encyklopedie.* [Online] Duben 8, 2010. [Cited: Duben 8, 2010.] http://cs.wikipedia.org/wiki/SQLite.

[7] RSS. *Wikipedie, otevřená encyklopedie.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://cs.wikipedia.org/wiki/RSS.

[8] **Kožina, Stanislav.** RSS a syndikovaný obsah. [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.kiv.zcu.cz/~brada/vyuka/files/pia/ppp/rss/.

[9] **Colby, Paul.** Calculating CPU Usage from /proc/stat. *PC Thoughts.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://colby.id.au/node/39.

[10] daemon-functions.sh. [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://blog.apokalyptik.com/files/bash-daemon/daemon-functions.sh.

[11] Java ME. *Wikipedie, otevřená encyklopedie.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://cs.wikipedia.org/wiki/Java_ME#Profily_spadaj.C3.ADc.C3.AD_pod_CLDC.

[12] .NET Compact Framework. *.NET Compact Framework.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://msdn.microsoft.com/en-us/netframework/aa497273.aspx.

[13] Testing for and Responding to Network Connections in the .NET Compact Framework. *MSDN.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://msdn.microsoft.com/en-us/library/aa446548.aspx.

[14] SQL Server Compact 3.5. *Microsoft SQL Server Compact 3.5.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.microsoft.com/sqlserver/2005/en/us/Compact.aspx.

[15] SQL Server Compact. *Wikipedie, otevřená encyklopedie.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://en.wikipedia.org/wiki/SQL_Server_Compact.

[16] HttpWebRequest and Ignoring SSL Certificate Errors . *Rick Strahl's Web Log.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.west-wind.com/weblog/posts/48909.aspx.

[17] FIX: A System.Net.WebException occurs when you run an application to send HTTPS Web requests to a server in an embedded device. *Technická podpora Microsoft.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://support.microsoft.com/kb/970549.

[18] Feedicons 2. *Zeusbox Studio.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.zeusboxstudio.com/blog/feedicons-2.

[19] Feed Icons - Home of the free Standard RSS Icon. *FeedIcons.com.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://www.feedicons.com/.

[20] Atom. *Wikipedie, otevřená encyklopedie.* [Online] Duben 7, 2010. [Cited: Duben 7, 2010.] http://cs.wikipedia.org/wiki/Atom_%28standard%29.

Příloha A

Zkratka RSS má následující významy:

- Rich Site Summary - stručný přehled obsahu webu
- Really Simple Syndication - opravdu jednoduchá syndikace obsahu

V roce 1999 firma Netscape definovala RSS dokument verze 0.90, šlo o prostý XML soubor s RDF hlavičkou. V roce 2000 byla uvolněna oficiální verze 0.91 RSS specifikace. V současné době je aktuální verze RSS 2.0, pro kterou byla vydána v roce 2003 specifikace. RSS dokument je založen na klasickém XML a musí tedy dodržovat specifikaci 1.0 XML. Jedná se o samopopisnou strukturu.

Ikona upozorňující na poskytování informací technologií RSS na webových sídlech:

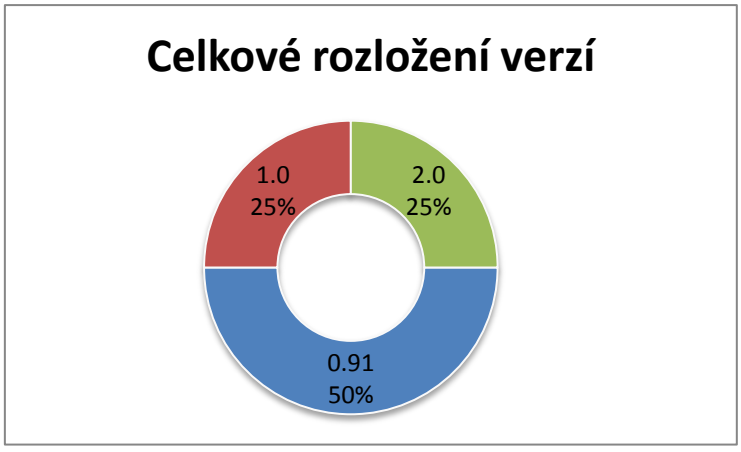
Obrázek 11

Verze RSS: 0.90, 0.91, 0.92, 1.0, 2.0

Přehled použití formátu verzí RSS dokumentu na webových sídlech:

Tabulka 8

Verze RSS dokumentu	Celkové rozložení verzí
0.91	50
1.0	25
2.0	25



Obrázek 12

V době psaní této práce nebyl definován standard pro žádnou z verzí RSS.

Struktura RSS dokumentu verze 2.0

Dokument začíná XML prologem - verzí použitého XML, uvádí také použitou znakovou sadu dokumentu a jedná se o povinný nepárový tag.

```
<?xml version="1.0" encoding="utf-8"?>
```

Element <rss>

Element nejvyšší úrovně - kořenový element. Uvádí verzi RSS, která je umístěna v atributu "version" a obsahuje jediný povinný element <channel>.

```
<rss version="2.0">
```

Element <channel>

Samotný element "kanál", který obsahuje další uzlové objekty a informace. Následující tabulka zobrazuje elementy, které se neopakují, a většina zde uvedených není vyžadována.

Tabulka 9

Element	Popis	Povinný element
<category>	Definuje kategorii informačního kanálu	N
<cloud>	Zajišťuje okamžitou notifikaci pro zaregistrované procesy v okamžiku aktualizace informačního kanálu	N
<copyright>	Práva na poskytovaná data	N
<description>	Popis informačního zdroje	A
<docs>	Udává URL adresu dokumentu popisujícího formát použitého při tvorbě RSS	N
<generator>	Popis generátoru sestavujícího RSS dokument	N
<image>	Element obsahující další podelementy sloužící k definování obrázku RSS informačního kanálu	N
<language>	Definuje jazyk, ve kterém je informační kanál napsán	N
<lastBuildDate>	Definuje datum poslední změny obsahu dokumentu	N
<link>	Odkaz na umístění RSS dokumentu	A
<managingEditor>	Definuje e-mailovou adresu tvůrce RSS dokumentu	N
<pubDate>	Definuje datum poslední publikace obsahu informačního kanálu	N
<rating>	Obrázek zobrazující hodnocení obsahu	N
<skipDays>	Udává, kolik dnů můžou agregační služby vynechat při aktualizaci	N
<skipHours>	Nastavení se týká agregátorů obsahu a udává, kolik hodin má být vynecháno při aktualizaci	N
<textInput>	Zobrazí textový výstup, který může být zadán	N
<title>	Název informačního kanálu	A
<ttl>	Udává dobu, po kterou vizualizér může držet načtené informace z kanálu v paměti, než provede aktualizaci dat	N
<webMaster>	E-mail na webmástra informačního kanálu	N

Element <item>

Element tvoří obsahovou část RSS dokumentu, tento element je povinný a musí být obsažen v elementu <channel> alespoň jednou.

Element	Popis	Povinný element
<author>	Udává e-mailovou adresu autora položky	N

<category>	Definuje kategorii, do které informace spadá	N
<comments>	Tento element udává odkaz na umístění komentářů vztahujících se k informačnímu kanálu	N
<description>	Povinný element, který nese samotnou informaci	A
<enclosure>	Definuje přítomnost multimédiálních souborů	N
<guid>	Jedinečný identifikátor položky	N
<link>	Odkaz na detail položky	A
<pubDate>	Definuje datum publikování položky	N
<source>	Specifikuje další zdroj informace	N
<title>	Název položky	A

Další obdobné technologie jsou:

- Atom 1.0 je přímým nástupcem RSS. Formát dokumentů typu ATOM se dočkal standardizace v prosinci 2005 a je popsán v dokumentu RFC 4287 [20].
- CDF

Ukázka struktury dokumentu RSS 2.0

Následuje úryvek validního RSS dokumentu vygenerovaného pomocí projektu „sysinfo2rss“, který je předmětem této práce.

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
 <channel>
 <title>feinuke system state info channel</title>
 <link>http://feinuke.vsb.cz/sysinfo2rss/rss.xml</link>
 <description>feinuke system state info channel</description>
 <category>Server monitoring</category>
 <image>
 <title>feinuke system state info channel</title>
 <link>http://feinuke.vsb.cz/sysinfo2rss/rss.xml</link>
 <url>http://feinuke.vsb.cz/sysinfo2rss/img/tux.png</url>
 </image>
 <atom:link href="http://feinuke.vsb.cz/sysinfo2rss/rss.xml" rel="self"
type="application/rss+xml" />
 <language>cs</language>
 <pubDate>Tue, 06 Apr 2010 02:02:01 +0200</pubDate>
 <lastBuildDate>Tue, 06 Apr 2010 02:02:01 +0200</lastBuildDate>
 <ttl>60</ttl>
<item>
<title>feinuke system status</title>
  <link>http://feinuke.vsb.cz/sysinfo2rss/sysinfo2rss.php?item=262</link>
  <pubDate>Tue, 06 Apr 2010 02:01:01 +0200</pubDate>
  <description><![CDATA[Uptime: 229d 15h 9m <br />
 CPU utilization: 0% <br />
  Used space on volume /data 4532MB <br />
 Available space on volume /data 93644MB <br />
 Usage volume /data 5% <br />
 Used space on volume /root 489MB <br />
 Available space on volume /root 2185MB <br />
 Usage volume /root 19% <br />
 Used space on volume /var 492MB <br />
 Available space on volume /var 56728MB <br />
 Usage volume /var 1% <br />
 Total memory 3931MB <br />
```

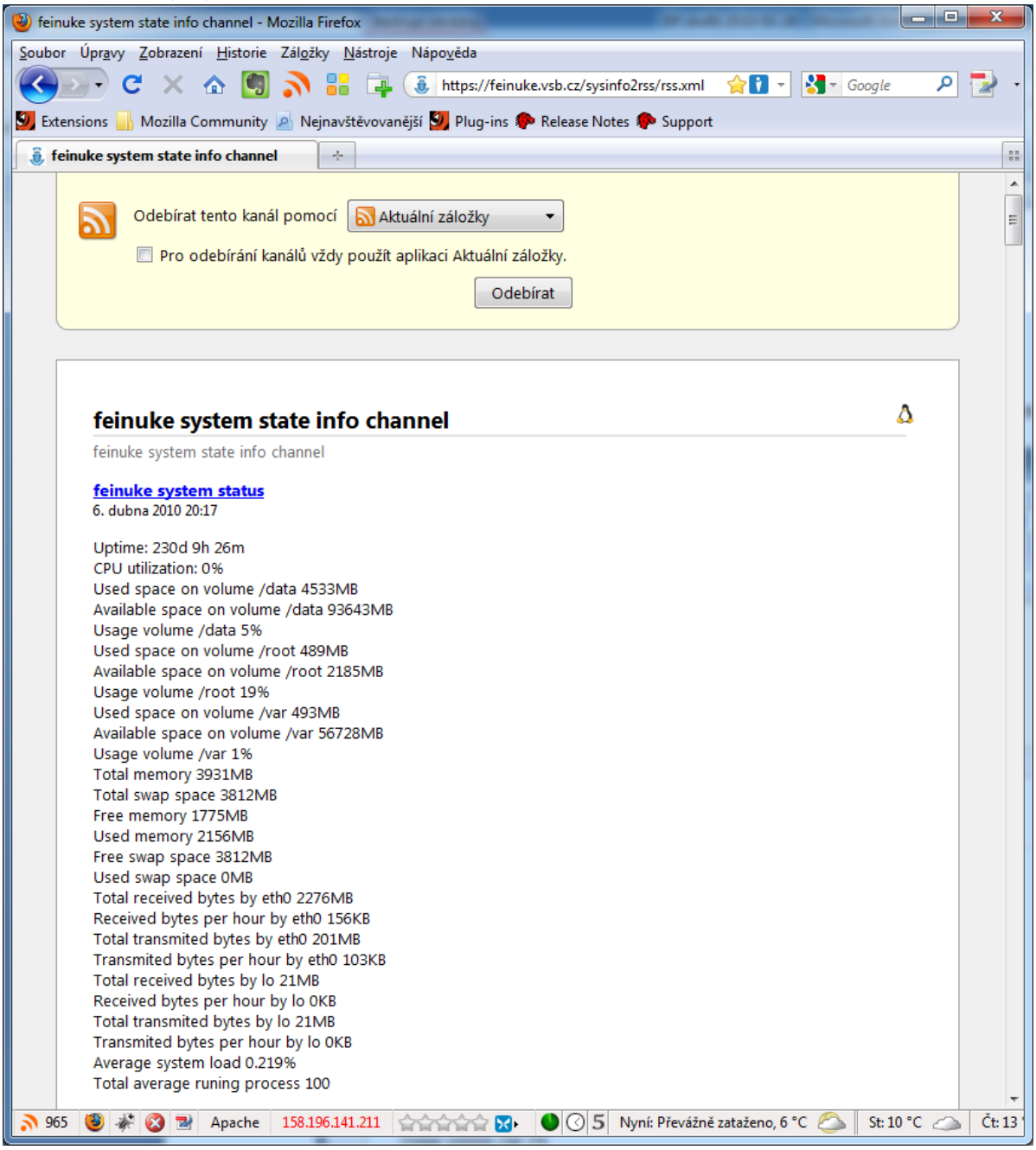
```
 Total swap space 3812MB <br />
 Free memory 1847MB <br />
 Used memory 2083MB <br />
 Free swap space 3812MB <br />
 Used swap space 0MB <br />
  Total received bytes by eth0 2271MB <br />
 Received bytes per hour by eth0 164KB <br />
 Total transmited bytes by eth0 170MB <br />
 Transmited bytes per hour by eth0 132KB <br />
 Total received bytes by lo 21MB <br />
 Received bytes per hour by lo 0KB <br />
 Total transmited bytes by lo 21MB <br />
 Transmited bytes per hour by lo 0KB <br />
 Average system load 0.233% <br />
 Total average runing process 99 <br />
]]></description>
  <guid isPermaLink="false"
>http://feinuke.vsb.cz/sysinfo2rss/sysinfo2rss.php?item=262</guid>
</item>
<item>
<title>feinuke system status</title>
  <link>http://feinuke.vsb.cz/sysinfo2rss/sysinfo2rss.php?item=261</link>
  <pubDate>Tue, 06 Apr 2010 01:00:04 +0200</pubDate>
  <description><![CDATA[Uptime: 229d 14h 8m <br />
 CPU utilization: 0% <br />
  Used space on volume /data 4532MB <br />
 Available space on volume /data 93644MB <br />
 Usage volume /data 5% <br />
 Used space on volume /root 489MB <br />
 Available space on volume /root 2185MB <br />
 Usage volume /root 19% <br />
 Used space on volume /var 492MB <br />
 Available space on volume /var 56728MB <br />
 Usage volume /var 1% <br />
 Total memory 3931MB <br />
 Total swap space 3812MB <br />
 Free memory 1850MB <br />
 Used memory 2081MB <br />
 Free swap space 3812MB <br />
 Used swap space 0MB <br />
  Total received bytes by eth0 2271MB <br />
 Received bytes per hour by eth0 221KB <br />
 Total transmited bytes by eth0 170MB <br />
 Transmited bytes per hour by eth0 223KB <br />
 Total received bytes by lo 21MB <br />
 Received bytes per hour by lo 0KB <br />
 Total transmited bytes by lo 21MB <br />
 Transmited bytes per hour by lo 0KB <br />
 Average system load 0.236% <br />
 Total average runing process 98 <br />
]]></description>
  <guid isPermaLink="false"
>http://feinuke.vsb.cz/sysinfo2rss/sysinfo2rss.php?item=261</guid>
</item>
...
</channel>
</rss>
```

Příloha B

Následující obrázky dokládají univerzálnost a obrovskou variabilitu v možnostech použití RSS technologie napříč různými platformami OS při vizualizaci pořízených dat. Výčet všech čteček a operačních systémů není konečný a vzhledem k omezenému místu zde uvádím jen malou část z nich.

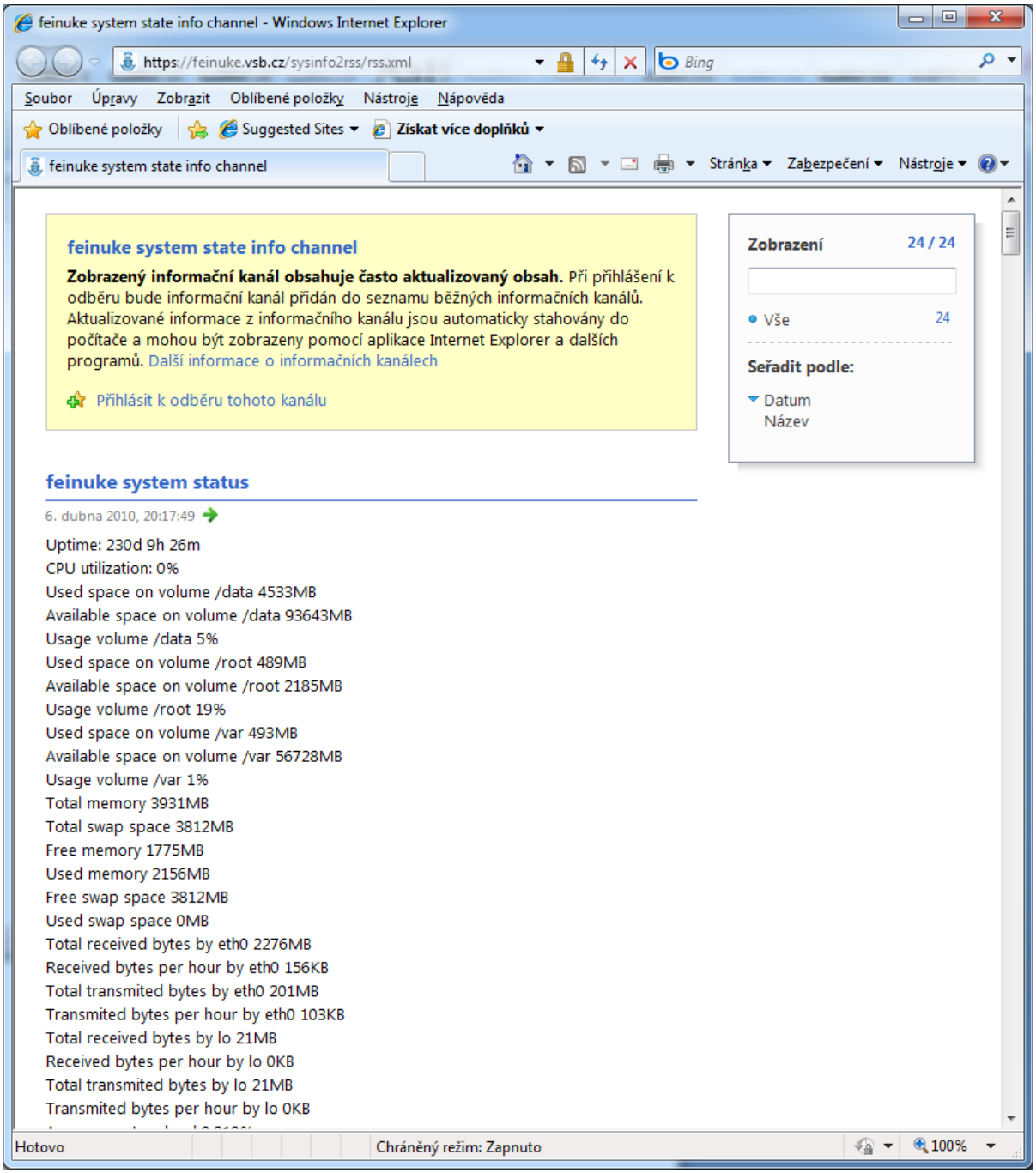
Integrované čtečky webových prohlížečů na platformě OS Microsoft Windows:

Mozilla Firefox (3.6)



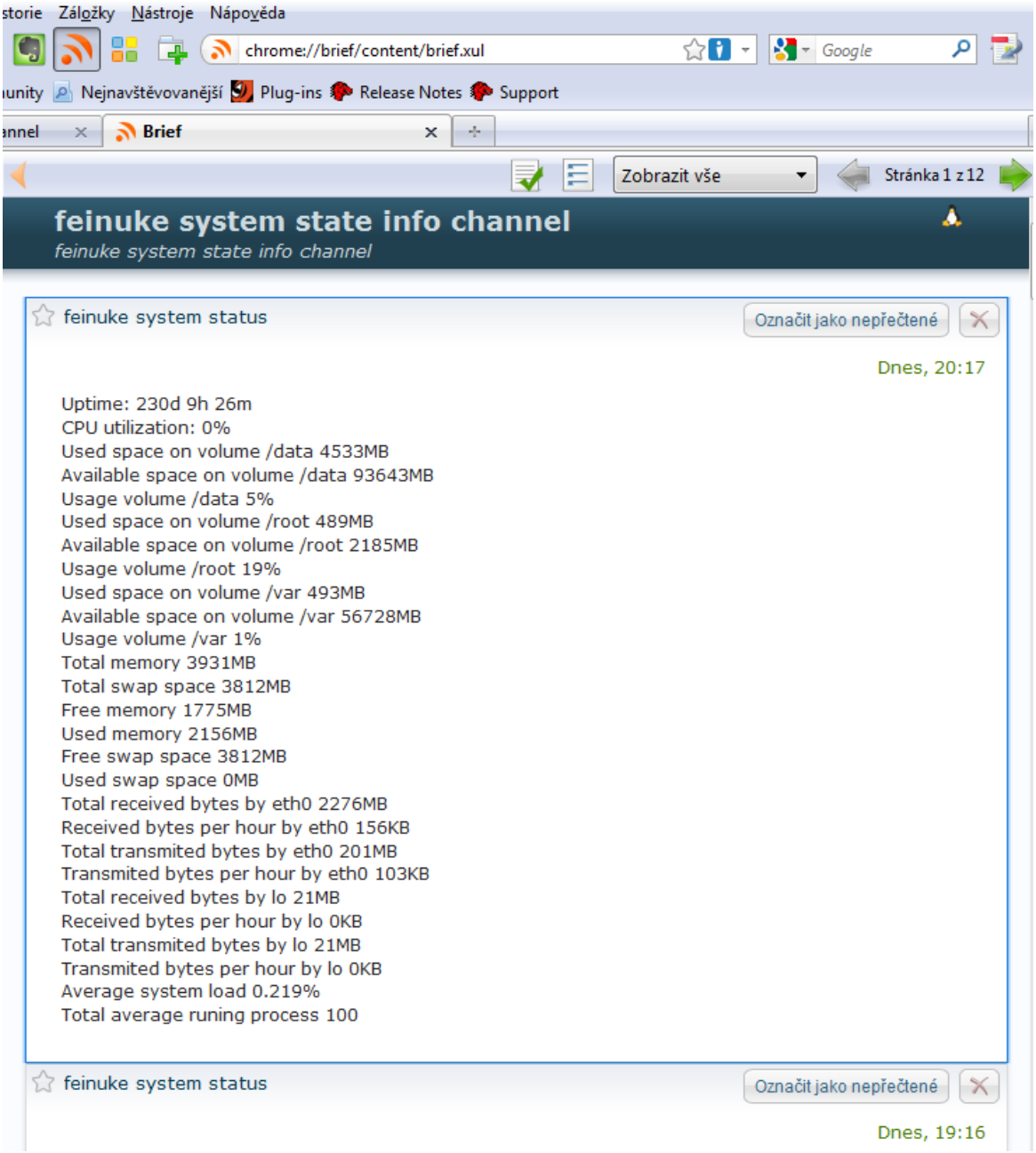
Obrázek 13

Internet Explorer (8.0)



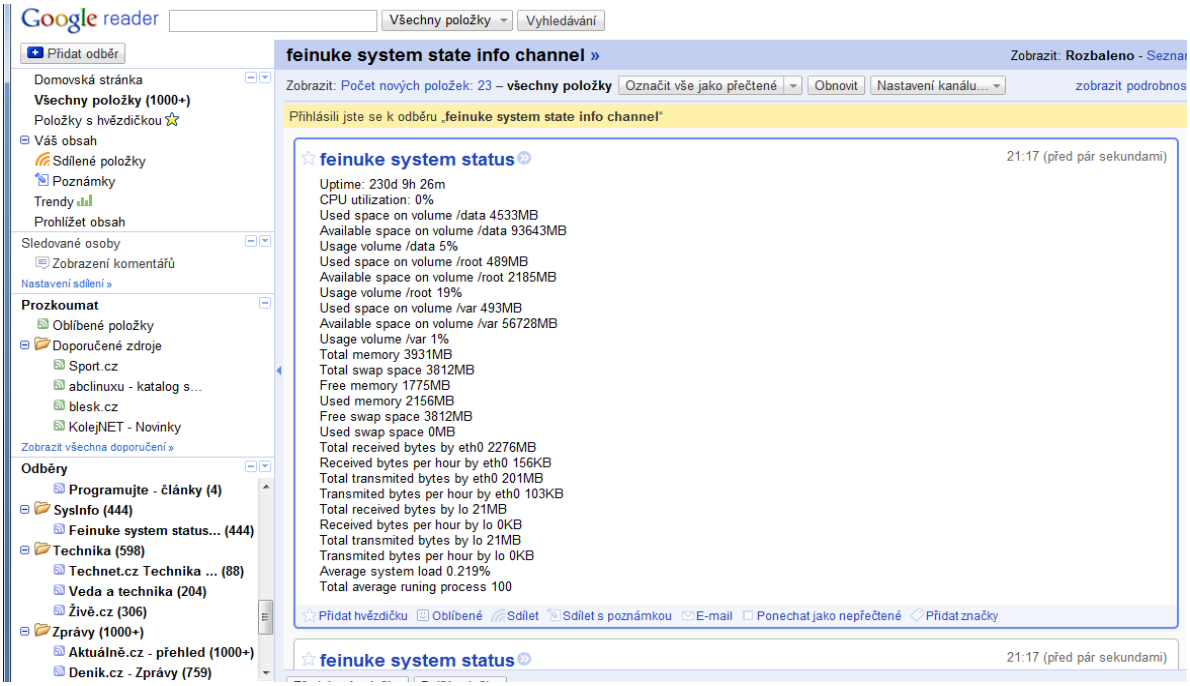
Obrázek 14

Rss čtečka jako doplněk Brief pro prohlížeč Mozilla Firefox



Obrázek 15

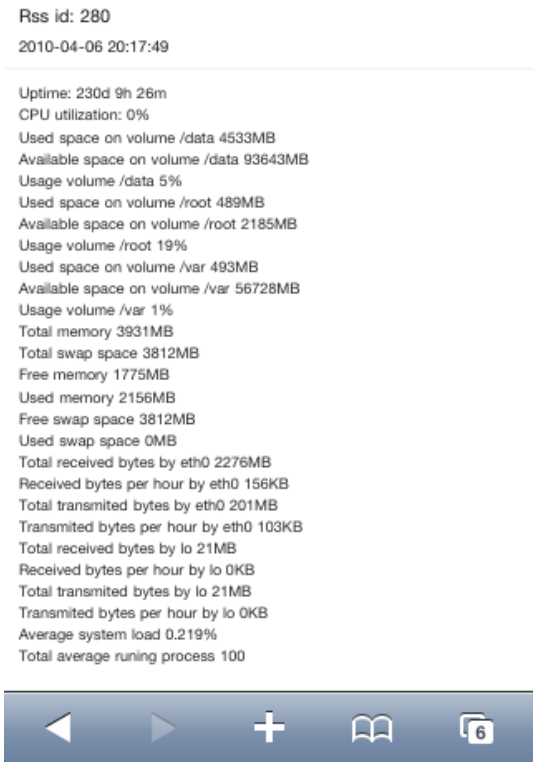
Webová služba Google Reader



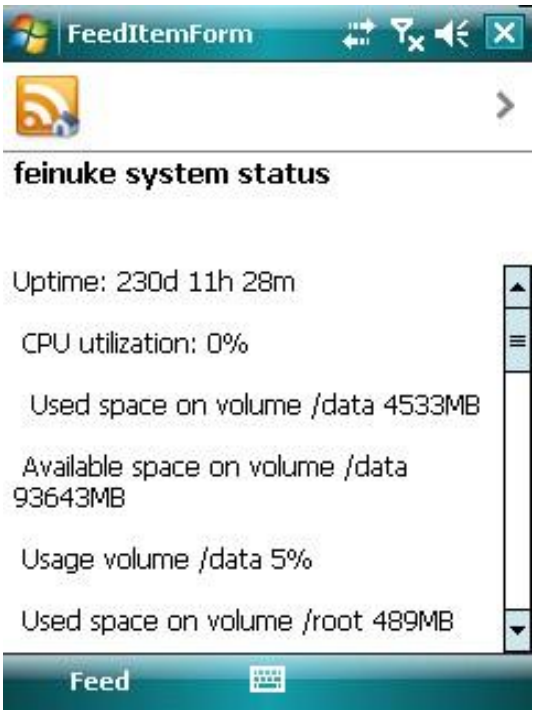
Obrázek 16

RSS čtečka na platformě iPhone OS 3.0

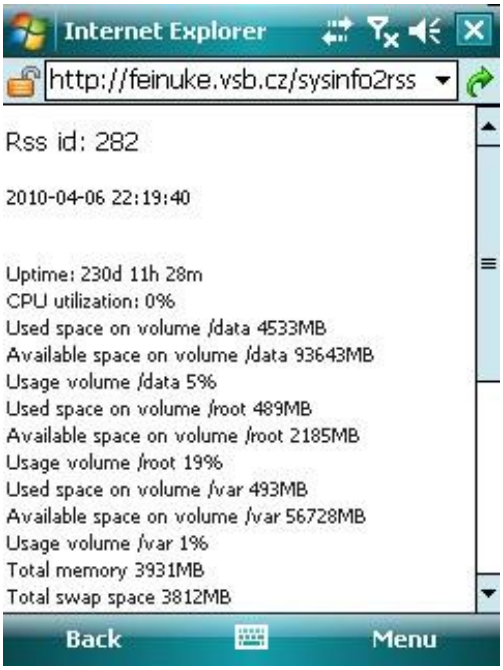
Webový prohlížeč Safari na platformě iPhone OS 3.0, zobrazující detail článku RSS dokumentu generovaného monitorovací aplikací „sysinfo2rss“



Detail položky RSS kanálu z aplikace „SysInfoRssReader“



Webový prohlížeč Internet Explorer na platformě Windows Mobile 6.0, zobrazující detail článku RSS dokumentu generovaného monitorovací aplikací „sysinfo2rss“



Příloha C

Třída MainForm

```
  1:  using System;
  2:  using System.ComponentModel;
  3:  using System.Drawing;
  4:  using System.Net;
  5:  using System.Threading;
  6:  using System.Windows.Forms;
  7:  using Microsoft.WindowsMobile.Status;
  8:
  9:  ///<summary>
 10:  /// Hlavni formular, ktery inicializuje dalsi formulare + udruzje reference
na vsechny objekty
 11:  ///</summary>
 12:
 13:  namespace SysInfoRssReader
 14:  {
 15:      public partial class MainForm : Form
 16:      {
 17:          private readonly Program _program;
 18:          private FeedStoreDb _feedStoreDb;
 19:          private Feeds _feeds;
 20:          private SystemState _connectionsCount;
 21:          private bool _connected;
 22:          private Color _textColor, _backgroundColor;
 23:
 24:          public MainForm(Program program)
 25:          {
 26:              InitializeComponent();
 27:              _program = program;
 28:
 29:              if (_program.InfoForm == null)
 30:              {
 31:                  _program.InfoForm = new InfoForm(_program);
 32:              }
 33:
 34:              _feedStoreDb = new FeedStoreDb();
 35:
 36:              _feeds = new Feeds();
 37:
 38:              _connectionsCount = new
SystemState(SystemProperty.ConnectionsCount);
 39:              _connectionsCount.Changed += ConnectionsCountChanged;
 40:              _textColor = Color.DodgerBlue;
 41:              _backgroundColor = Color.FromArgb(240, 240, 240);
 42:          }
 43:
 44:          #region Getters/Setters
 45:          //instance tridy manipulujici s databazi
 46:          public FeedStoreDb GetFeedStore
 47:          {
 48:              get { return _feedStoreDb; }
 49:          }
 50:
 51:          //instance tridy udrzujici reference na objekty typu kategorie
 52:          public Feeds GetFeeds
 53:          {
 54:              get { return _feeds; }
 55:          }
 56:
 57:          //vraci true, pokud je detekovano pripojeni
```

```
 58:          public bool GetConnected
 59:          {
 60:              get { return _connected; }
 61:          }
 62:
 63:          //vrati barvu textu
 64:          public Color GetTextColor
 65:          {
 66:              get { return _textColor; }
 67:          }
 68:
 69:          //vrati barvu pozadi
 70:          public Color GetBackgroundColor
 71:          {
 72:              get { return _backgroundColor; }
 73:          }
 74:          #endregion
 75:
 76:          #region Menu -> Close exit from MainApp
 77:          private void menuItem1_Click(object sender, EventArgs e)
 78:          {
 79:              CloseMainApp();
 80:          }
 81:          #endregion
 82:
 83:          #region Menu -> Help -> AboutApp
 84:          private void menuItem5_Click(object sender, EventArgs e)
 85:          {
 86:              if (_program.AboutForm == null)
 87:              {
 88:                  _program.AboutForm = new AboutForm(_program);
 89:              }
 90:
 91:              _program.AboutForm.Show();
 92:          }
 93:          #endregion
 94:
 95:          #region Menu -> Help -> Help
 96:          private void menuItem4_Click(object sender, EventArgs e)
 97:          {
 98:              Help.ShowHelp(this, @"\windows\SysInfoRssReaderHelp.htm#obsah");
 99:          }
100:          #endregion
101:
102:          #region Menu -> Settings -> Add warnings
103:          private void menuItem8_Click(object sender, EventArgs e)
104:          {
105:              if (_program.AddWarningsForm == null)
106:              {
107:                  _program.AddWarningsForm = new AddWarningsForm(_program);
108:              }
109:
110:              _program.AddWarningsForm.Show();
111:              _program.AddWarningsForm.LoadFeedList();
112:          }
113:          #endregion
114:
115:          #region Menu -> Seettings -> Delete warnings
116:          private void menuItem9_Click(object sender, EventArgs e)
117:          {
118:              if (_program.DelWarningsForm == null)
119:              {
120:                  _program.DelWarningsForm = new DelWarningsForm(_program);
```

```
121:             }
122:
123:             _program.DelWarningsForm.Show();
124:             _program.DelWarningsForm.LoadFeedList();
125:         }
126:         #endregion
127:
128:             //string newLine = Environment.NewLine;//novy radek do textboxu
129:
130:         #region Close main aplication
131:         private void MainForm_Closing(object sender, CancelEventArgs e)
132:         {
133:             CloseMainApp();
134:         }
135:         #endregion
136:
137:         #region Loading MainForm
138:         private void MainForm_Load(object sender, EventArgs e)
139:         {
140:             OverSpojeniAsync();
141:
142:             if (_program.UpdateInfoForm == null)
143:             {
144:                 _program.UpdateInfoForm = new UpdateInfoForm(_program);
145:             }
146:         }
147:         #endregion
148:
149:         #region AddFeedForm
150:         /// <summary>
151:         /// fce zajisti zobrazeni formulare pro pidani
feedu(channel)/kategorie
152:         /// </summary>
153:         /// <param name="sender"></param>
154:         /// <param name="e"></param>
155:         private void pictureBox1_Click(object sender, EventArgs e)
156:         {
157:             if (_program.AddFeedForm == null)
158:             {
159:                 _program.AddFeedForm = new AddFeedForm(_program);
160:             }
161:
162:             _program.AddFeedForm.Show();
163:         }
164:
165:         private void linkLabel1_Click(object sender, EventArgs e)
166:         {
167:             pictureBox1_Click(sender, e);
168:         }
169:         #endregion AddFeedForm
170:
171:         #region DeleteFeedForm
172:         /// <summary>
173:         /// fce smaze feedy/kategorie
174:         /// </summary>
175:         /// <param name="sender"></param>
176:         /// <param name="e"></param>
177:         private void pictureBox2_Click(object sender, EventArgs e)
178:         {
179:             if (_program.DeleteFeedForm == null)
180:             {
181:                 _program.DeleteFeedForm = new DeleteFeedForm(_program);
182:             }
```

```
183:
184:                 _program.DeleteFeedForm.Show();//spustime jako prvni metodu show,
ta zajisti inicializaci vsech prvku, pak muzeme zajistit jejich vykresleni pres
updatepanel metodu
185:                 _program.DeleteFeedForm.UpdatePanelData();
186:             }
187:
188:             private void linkLabel2_Click(object sender, EventArgs e)
189:             {
190:                 pictureBox2_Click(sender, e);
191:             }
192:             #endregion
193:
194:             #region Update feeds
195:             /// <summary>
196:             /// fce spusti nacteni dat z internetu, parsovani dat, pridani dat do
databaze a naplneni prislusnych objektu daty
197:             /// </summary>
198:             /// <param name="sender"></param>
199:             /// <param name="e"></param>
200:             private void pictureBox3_Click(object sender, EventArgs e)
201:             {
202:                 if (GetConnected)
203:                 {
204:                     _program.UpdateInfoForm.Show();
205:                     _program.UpdateInfoForm.UpdatePanelData();
206:                     Thread downloadThred = new Thread(UpdateFeeds);
207:                     downloadThred.Start();
208:                 }
209:                 else
210:                 {
211:                     _program.UpdateInfoForm.Show();
212:                     _program.UpdateInfoForm.DrawPanelNoConnection();
213:                 }
214:             }
215:
216:             private void linkLabe3_Click(object sender, EventArgs e)
217:             {
218:                 pictureBox3_Click(sender, e);
219:             }
220:             #endregion
221:
222:             #region FeedsForm
223:             /// <summary>
224:             /// fce zobrazime kanaly a kategori
225:             /// </summary>
226:             /// <param name="sender"></param>
227:             /// <param name="e"></param>
228:             private void pictureBox4_Click(object sender, EventArgs e)
229:             {
230:                 if (_program.FeedsForm == null)
231:                 {
232:                     _program.FeedsForm = new FeedsForm(_program);
233:                 }
234:
235:                 _program.FeedsForm.Show();
236:
237:                 if (!_program.FeedsForm.Controls.Equals(null))
238:                 {
239:                     _program.FeedsForm.UpdatePanelData();
240:                 }
241:
242:             }
```

```
243:          private void linkLabe4_Click(object sender, EventArgs e)
244:          {
245:              pictureBox4_Click(sender, e);
246:          }
247:          #endregion
248:
249:          #region AboutForm
250:          /// <summary>
251:          /// fce zobrazi formular o aplikaci
252:          /// </summary>
253:          /// <param name="sender"></param>
254:          /// <param name="e"></param>
255:          private void pictureBox5_Click(object sender, EventArgs e)
256:          {
257:              if (_program.AboutForm == null)
258:              {
259:                  _program.AboutForm = new AboutForm(_program);
260:              }
261:              _program.AboutForm.Show();
262:
263:          }
264:          private void linkLabe5_Click(object sender, EventArgs e)
265:          {
266:              pictureBox5_Click(sender, e);
267:          }
268:          #endregion
269:
270:          //btn - ukonceni aplikace
271:          private void pictureBox6_Click(object sender, EventArgs e)
272:          {
273:              CloseMainApp();
274:          }
275:
276:          /// <summary>
277:          /// fce ukonci aplikaci
278:          /// </summary>
279:          private void CloseMainApp()
280:          {
281:              //zavreme spojeni na DB
282:              if (_feedStoreDb.GetSCon != null)
283:              {
284:                  _feedStoreDb.DbCloseCon();
285:              }
286:
287:              //ukoncime aplikaci
288:              Close();
289:          }
290:
291:          /// <summary>
292:          /// fce projde vsechny kategorie a feedy(chnnel), pro kazdy feed
stahne xml soubor, ktery rozparsuje na item polozky
293:          /// </summary>
294:          public void UpdateFeeds()
295:          {
296:              string xmlData;
297:              _program.UpdateInfoForm.UpdateEventLabel("Downloading data for:
");
298:              //projdeme kategori a pro vsechny feedy...
299:              foreach (Category catItem in
_program.GetMainForm.GetFeeds.CategoryFeeds)
300:              {
301:                  foreach (Feed feedItem in catItem.CatFeed)
302:                  {
```

```
303:                        _program.UpdateInfoForm.UpdateFeedLabel(feedItem, false);
304:                        xmlData = Downloader.DownloadFeedData(feedItem.Link,
_program); //stahne data
305:
306:                        if (!xmlData.Equals(""))
307:                        {
308:                            ParseRss.ParseFeedItemData(xmlData, feedItem,
_program);//stazena data rozparsuje
309:                            _feedStoreDb.SaveFeedItem(feedItem);//ulozi
feed(channel) do db
310:                        }
311:                        _program.UpdateInfoForm.UpdateFeedLabel(feedItem, true);
312:                    }
313:                }
314:                 _program.UpdateInfoForm.UpdateEventLabel("Downloading data is
done");
315:            }
316:
317:            //delagat pro testovani pripojeni
318:            void ConnectionsCountChanged(object sender, ChangeEventArgs args)
319:            {
320:                OverSpojeniAsync();
321:            }
322:
323:            //vlakno pro spusteni testu pripojeni
324:            public void OverSpojeniAsync()
325:            {
326:                Thread t = new Thread(GetConnection);
327:                t.Start();
328:            }
329:
330:            //pokud ma adapter pridelenou adresu jinou nez 127.0.0.1 vraci true
331:            public void GetConnection()
332:            {
333:                try
334:                {
335:                    string hostName = Dns.GetHostName();
336:                    IPHostEntry hostEntry = Dns.GetHostEntry(hostName);
337:                    string hostIpAdd = hostEntry.AddressList[0].ToString();
//zjisti prvni IP ze vsech spojeni
338:                    _connected = (hostIpAdd !=
IPAddress.Parse("127.0.0.1").ToString());
339:                }
340:                catch (Exception e)
341:                {
342:                    string x = e.Message;
343:                }
344:            }
345:        }
346:    }
```

Třída Downloader

```
  1:  namespace SysInfoRssReader
  2:  {
  3:      public static class Downloader
  4:      {
  5:          static Downloader()
  6:          {
  7:              ServicePointManager.CertificatePolicy = new
AcceptAllCertificatePolicy();
  8:          }
  9:
```

```
 10:          /// <summary>
 11:          /// fce stahne ze zadane url vzdaleny dokument
 12:          /// </summary>
 13:          /// <param name="url">string</param>
 14:          /// <param name="program">Program</param>
 15:          /// <returns>string - xml dokument</returns>
 16:          public static string DownloadFeedData(string url, Program program)
 17:          {
 18:              HttpWebResponse httpWebResponse = null;
 19:              StreamReader reader = null;
 20:              string content = "";
 21:
 22:              try
 23:              {
 24:                  HttpWebRequest httpWebRequest =
(HttpWebRequest)WebRequest.Create(url);
 25:                  httpWebRequest.Method = "GET";
 26:                  httpWebRequest.UserAgent = ".NETCF HttpWebRequest agent";
 27:                  httpWebRequest.Timeout = 10000;
 28:                  httpWebRequest.KeepAlive = true;
 29:
 30:                  httpWebResponse =
(HttpWebResponse)httpWebRequest.GetResponse();
 31:                  Stream stream = httpWebResponse.GetResponseStream();
 32:                  reader = new StreamReader(stream);
 33:                  //nacteni textu odpovedi
 34:                  content = reader.ReadToEnd();
 35:
 36:              }
 37:              catch (Exception ex)
 38:              {
 39:                  string errMsg = "Doslo k vyjimce: " + ex + " pri stahovani
dat z url: " + url;
 40:                  program.InfoForm.Show();
 41:                  program.InfoForm.SetMsg("Error", errMsg,
program.GetMainForm);
 42:
 43:              }
 44:              finally
 45:              {
 46:                  if (reader != null)
 47:                  {
 48:                      reader.Close();
 49:                  }
 50:                  if (httpWebResponse != null)
 51:                  {
 52:                      httpWebResponse.Close();
 53:                  }
 54:              }
 55:
 56:              return content;
 57:          }
 58:
 59:          public static Bitmap DownloadImage(string url, Program program)
 60:          {
 61:              Bitmap image = null;
 62:              HttpWebResponse httpWebResponse = null;
 63:              Stream stream = null;
 64:
 65:              try
 66:              {
 67:                  HttpWebRequest httpWebRequest =
(HttpWebRequest)WebRequest.Create(url);
```

```
 68:                   ServicePointManager.CertificatePolicy = new
AcceptAllCertificatePolicy();
 69:                   httpWebRequest.Method = "GET";
 70:                   httpWebRequest.UserAgent = ".NETCF HttpWebRequest agent";
 71:                   httpWebRequest.Timeout = 10000;
 72:                   httpWebRequest.KeepAlive = true;
 73:
 74:                   httpWebResponse =
(HttpWebResponse)httpWebRequest.GetResponse();
 75:                   //ziskani datoveho proudu odpovedi
 76:                   stream = httpWebResponse.GetResponseStream();
 77:                   image = new Bitmap(stream);
 78:
 79:               }
 80:               catch (Exception ex)
 81:               {
 82:                   string errMsg = "Doslo k vyjimce: " + ex + " pri stahovani
dat z url: " + url;
 83:                   program.InfoForm.Show();
 84:                   program.InfoForm.SetMsg("Error", errMsg,
program.GetMainForm);
 85:               }
 86:               finally
 87:               {
 88:                   if (stream != null)
 89:                   {
 90:                       stream.Close();
 91:                   }
 92:                   if (httpWebResponse != null)
 93:                   {
 94:                       httpWebResponse.Close();
 95:                   }
 96:               }
 97:
 98:               return image;
 99:           }
100:   }
```

Třída AcceptAllCertificatePolicy

```
1:  internal class AcceptAllCertificatePolicy : ICertificatePolicy
2:  {
3:          public bool CheckValidationResult(ServicePoint sPoint, X509Certificate
cert, WebRequest wRequest, int certProb)
4:          {
5:              //certifikat je vzdy akceptovan
6:              return true;
7:          }
8:  }
```

Třídat ParseRss

```
  1:  using System;
  2:  using System.Linq;
  3:  using System.Collections.Generic;
  4:  using System.Xml.Linq;
  5:  using System.Drawing;
  6:
  7:  ///<summary>
  8:  /// Staticka trida parsuje RSS 2.0 xml dokument a naplnuje feed(channel)
objekty daty
  9:  ///</summary>
 10:
```

```
 11:  namespace SysInfoRssReader
 12:  {
 13:      public static class ParseRss
 14:      {
 15:          /// <summary>
 16:          /// fce rozparsuje RSS 2.0 xml dokument
 17:          /// </summary>
 18:          /// <param name="xmlData">xml dokument</param>
 19:          /// <returns>vrati objekt feed obsahujci data</returns>
 20:          public static Feed ParseFeedData(string xmlData, Program program)
 21:          {
 22:              List<Feed> feeds = null;
 23:
 24:              //zjistime jestli string obsahuje data
 25:              if (!string.IsNullOrEmpty(xmlData))
 26:              {
 27:                  //XmlTextReader xmlTextRader = new XmlTextReader(new
System.IO.StringReader(data));
 28:                  XDocument feedData = XDocument.Parse(xmlData);
 29:                  try
 30:                  {
 31:                      feeds = (from channel in feedData.Descendants("channel")
 32:                               select new Feed
 33:                               {
 34:                                   Title = ((string)channel.Element("title")),
 35:                                   //Link = ((string) channel.Element("link")),
 36:                                   PubDate =
(string)channel.Element("pubDate"),
 37:                                   IconLink =
(channel.Element("image").Element("url").Value),
 38:                                   ItemCount =
channel.Elements("item").Count(),
 39:                                   FeedItems = null,
 40:                               }).ToList();
 41:                  }
 42:                  catch (Exception ex)
 43:                  {
 44:                      string errMsg = "Doslo k vyjimce: " + ex + " pri
parsovani dat";
 45:                      program.InfoForm.Show();
 46:                      program.InfoForm.SetMsg("Error", errMsg,
program.GetMainForm);
 47:                  }
 48:              }
 49:
 50:              return feeds[0];
 51:          }
 52:
 53:          //zjistime url adresu obrazku kanalu
 54:          public static void ParseFeedItemData(string xmlData, Feed feed,
Program program)
 55:          {
 56:              List<Feed> feeds = null;
 57:
 58:              if (!string.IsNullOrEmpty(xmlData))
 59:              {
 60:                  {
 61:                      XDocument feedData = XDocument.Parse(xmlData);
 62:
 63:                      try
 64:                      {
 65:                          feeds = (from channel in
feedData.Descendants("channel")
```

```
 66:                                     select new Feed
 67:                                     {
 68:                                         FeedItems = (from item in
channel.Descendants("item")
 69:                                                      select new FeedItem
 70:                                                      {
 71:                                                          Title =
((string)item.Element("title")).Trim(),
 72:                                                          Description =
((string)item.Element("description")).Trim(),
 73:                                                          PubDate =
Utils.ParseDateToDbDate((string)item.Element("pubDate")),
 74:                                                          Link =
(string)item.Element("link"),
 75:                                                          Author =
(string)item.Element("author"),
 76:                                                          Warning =
Utils.FindPattern(feed.Warnings, ((string)item.Element("description")).Trim())
 77:                                                      }).ToList(),
 78:                                     }).ToList();
 79:                     }
 80:                     catch (Exception ex)
 81:                     {
 82:
 83:                         string errMsg = "Doslo k vyjimce: " + ex + " pri
parsovani dat";
 84:                         program.InfoForm.Show();
 85:                         program.InfoForm.SetMsg("Error", errMsg,
program.GetMainForm);
 86:                     }
 87:                 }
 88:             }
 89:             //prekopirovani feed item do feedu
 90:             feed.FeedItems = feeds[0].FeedItems;
 91:         }
 92:
 93:         //zkontroluje verzi rss
 94:         private static bool CheckRssVersion(string xmlData)
 95:         {
 96:             bool res = false;
 97:             string ver;
 98:
 99:             //pokud je xmlRssNode prazdny, nejedna se o validni RSS
100:             if (!string.IsNullOrEmpty(xmlData))
101:             {
102:                 XDocument feedData = XDocument.Parse(xmlData);
103:                 ver =
feedData.Element("rss").Attribute("version").ToString();
104:                 //kontrola verze - zatim umime pouze 2.0
105:                 if (ver.Equals(ver))
106:                 {
107:                     res = true;
108:                 }
109:             }
110:
111:             return res;
112:         }
113:     }
114: }
```

Třída FeedStoreDB

```
  1: using System;
```

```
  2:  using System.Collections;
  3:  using System.Collections.Generic;
  4:  using System.IO;
  5:  using System.Data;
  6:  using System.Data.SqlServerCe;
  7:
  8:  namespace SysInfoRssReader
  9:  {
 10:      public class FeedStoreDb
 11:      {
 12:          private string _conString, _dbPath, _dbFileName;
 13:          private SqlCeConnection sCon;
 14:          //private Program _program;
 15:
 16:          public FeedStoreDb()
 17:          {
 18:              //_program = program;
 19:              _dbPath =
Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().C
odeBase);//cesta k adresari kde je umistena aplikace
 20:              _dbFileName = @"\SysInfoRssReaderStore.sdf"; //nazev databazoveho
souboru
 21:              _conString = "Data Source=" + _dbPath + _dbFileName;
 22:              sCon = new SqlCeConnection(_conString);//sestaveni retezce pro
pripojeni
 23:
 24:              if (!File.Exists(_dbPath + _dbFileName))
 25:              {
 26:                  CreateFeedStoreDb();
 27:              }
 28:              else
 29:              {
 30:                  sCon.Open();
 31:              }
 32:          }
 33:
 34:          private void CreateFeedStoreDb()
 35:          {
 36:              SqlCeEngine engine = new SqlCeEngine();
 37:              engine.LocalConnectionString = _conString;//nastavit connection
string
 38:              engine.CreateDatabase();//vytvoreni DB souboru
 39:          }
 40:
 41:          //otevre spojeni s DB
 42:          public void DbOpenCon()
 43:          {
 44:              sCon.Open();
 45:          }
 46:
 47:          //Zavre pripojeni k DB
 48:          public void DbCloseCon()
 49:          {
 50:              sCon.Close();
 51:          }
 52:
 53:          //vrati objekt obsahujici pripojeni k databazi
 54:          public SqlCeConnection GetSCon
 55:          {
 56:              get {return sCon;}
 57:          }
 58:
 59:          /// <summary>
```

```
 60:          /// fce vlozi do tabulkek category a feedstore data
 61:          /// </summary>
 62:          /// <param name="feed">feed(channel)</param>
 63:          /// <param name="catName">jmeno kategorie</param>
 64:          public void AddFeed(Feed feed, String catName)
 65:          {
 66:              SqlCeCommand sComIns1, sComIns2, sComSel1, sComSel2;//sComIns3,
 67:              SqlCeDataReader reader1, reader2;
 68:              int lastCatAI;
 69:              bool exist;
 70:
 71:              sComSel1 = new SqlCeCommand("SELECT catId FROM category WHERE
catName = '" + catName + "'", sCon);
 72:              reader1 = sComSel1.ExecuteReader();
 73:
 74:              exist = reader1.Read();
 75:
 76:              //pokud neexistuje kategori se zadanym jmenem
 77:              if (!exist)
 78:              {
 79:                  sComIns2 = new SqlCeCommand("INSERT INTO category (catName)
VALUES('" + catName + "')", sCon); //vlozime novou kategorii do tb category
 80:                  sComIns2.ExecuteNonQuery();
 81:
 82:                  sComSel2 = new SqlCeCommand("SELECT max(catId) FROM
category", sCon);//zjistime id naposledy pridane kategorie
 83:                  reader2 = sComSel2.ExecuteReader();
 84:                  reader2.Read();
 85:
 86:                  //zjisti id posledniho vlozeneho zaznamu - nahrada za
autoinkrement
 87:                  lastCatAI = reader2.GetInt32(0);
 88:
 89:                  //sComIns1 = new SqlCeCommand("INSERT INTO feedstore (name,
url, icon, catId) VALUES ('" + feed.Name + "', '" + feed.Url + "', " + feed.Icon +
", " + lastCatAI + ")", sCon);
 90:                  //sComIns1.ExecuteNonQuery();
 91:
 92:                  //pridame novy feed do tb feedstore
 93:                  sComIns1 = new SqlCeCommand("INSERT INTO feedstore (title,
link, icon, catId)"+
 94:                      " VALUES (@title, @link, @icon, @lastCatAI)", sCon);
 95:
 96:                  sComIns1.Parameters.Add("@title", SqlDbType.NVarChar).Value =
feed.Title;
 97:                  sComIns1.Parameters.Add("@link", SqlDbType.NVarChar).Value =
feed.Link;
 98:                  sComIns1.Parameters.Add("@icon", SqlDbType.Image).Value =
Utils.ImageToByteArray(feed.Icon);
 99:                  sComIns1.Parameters.Add("@lastCatAI", SqlDbType.Int).Value =
lastCatAI;
100:                  sComIns1.ExecuteNonQuery();
101:
102:                  sComSel2 = new SqlCeCommand("SELECT max(id) FROM feedstore",
sCon);//zjistime id naposledy pridaneho feedu
103:                  reader2 = sComSel2.ExecuteReader();
104:                  reader2.Read();
105:              }
106:              else
107:              {
108:                  lastCatAI = reader1.GetInt32(0);
109:
```

```
110:                    sComIns1 = new SqlCeCommand("INSERT INTO feedstore (title,
link, icon, catId)" +
111:                       " VALUES (@title, @link, @icon, @lastCatAI)", sCon);
112:
113:                    sComIns1.Parameters.Add("@title", SqlDbType.NVarChar).Value =
feed.Title;
114:                    sComIns1.Parameters.Add("@link", SqlDbType.NVarChar).Value =
feed.Link;
115:                    sComIns1.Parameters.Add("@icon", SqlDbType.Image).Value =
Utils.ImageToByteArray(feed.Icon);
116:                    sComIns1.Parameters.Add("@lastCatAI", SqlDbType.Int).Value =
lastCatAI;
117:                    sComIns1.ExecuteNonQuery();
118:
119:                    sComSel2 = new SqlCeCommand("SELECT max(id) FROM feedstore",
sCon);//zjistime id naposledy pridaneho feedu
120:                    reader2 = sComSel2.ExecuteReader();
121:                    reader2.Read();
122:                    //lastFeedAI = reader2.GetInt32(0);
123:                    //pridame zaznam o vazbe mezi feedem a kategorri do vazebni
tb cat
124:                    //sComIns3 = new SqlCeCommand("INSERT INTO cat (catId,
feedId) VALUES (" + lastCatAI + ", " + lastFeedAI + ")", sCon);
125:                    //sComIns3.ExecuteNonQuery();
126:                }
127:            }
128:
129:            /// <summary>
130:            /// fce vytvori feed listu na zaklade dat z tabulek category,
feedstore, feeditem a warnings
131:            /// sestaveni parent objektu feeds->category->feed->item
132:            /// </summary>
133:            /// <param name="feeds">objekt feeds predstavuje List categorii,
ktere udrzuji reference na feedy(channel)</param>
134:            /// <returns>vraci objekt typu Feeds</returns>
135:            public Feeds LoadFeedsList(Feeds feeds)
136:            {
137:                SqlCeCommand sComSel1, sComSel3;
138:                SqlCeDataReader reader1, reader3;
139:                Category categoryFeeds;
140:
141:                if(feeds.CategoryFeeds.Count != 0)
142:                {
143:                    feeds.CategoryFeeds.Clear();
144:                }
145:
146:                sComSel1 = new SqlCeCommand("SELECT catId, catName FROM category
ORDER BY catName ASC", sCon);
147:
148:               reader1 = sComSel1.ExecuteReader();
149:
150:                //projdeme vsechny zaznamy pro kategorie
151:                while(reader1.Read())
152:                {
153:                    categoryFeeds = new Category(reader1.GetInt32(0),
reader1["catName"].ToString());//nova kategorie, nastavime nazev
154:
155:                    //najdeme feedy pro konkretni kategorii
156:                    sComSel3 = new SqlCeCommand("SELECT * FROM feedstore WHERE
catId = " + reader1.GetInt32(0) + " ORDER BY title ASC", sCon);
157:                    reader3 = sComSel3.ExecuteReader();
158:
159:                        //pridame do kategorie prislusne feedy
```

```
160:                        while(reader3.Read())
161:                        {
162:                            categoryFeeds.AddFeed((new Feed(reader3.GetInt32(0),
reader3["title"].ToString(), reader3["link"].ToString(), "0000-00-00",
163:
Utils.ByteArrayToImage((byte[])reader3["icon"]), reader3.GetInt32(2), 0,
LoadFeedItem(reader3.GetInt32(0)), LoadPatterns(reader3.GetInt32(0)))));
164:                            ///TODO: pridavani datumu, pocet polozek ve feedu
165:                        }
166:                    feeds.AddCategory(categoryFeeds);//vlozime categorii do top
objektu feeds
167:                }
168:                return feeds;
169:            }
170:
171:            /// <summary>
172:            /// fce nacte z tabulky feeditem item polozky
173:            /// </summary>
174:            /// <param name="feedId">id feedu (channel)</param>
175:            /// <returns>vraci List FeedItem</returns>
176:            private List<FeedItem> LoadFeedItem(int feedId)
177:            {
178:                List<FeedItem> feedItems = new List<FeedItem>();
179:
180:                SqlCeCommand sComSel1;
181:                SqlCeDataReader reader1;
182:
183:                sComSel1 = new SqlCeCommand("SELECT * FROM feeditem WHERE feedId
= " + feedId + " ORDER BY title ASC", sCon);
184:                reader1 = sComSel1.ExecuteReader();
185:
186:                while (reader1.Read())
187:                {
188:                    feedItems.Add(new FeedItem((string)reader1["title"],
(string)reader1["link"], (string)reader1["description"], (int)reader1["isread"],
(int)reader1["warning"]));
189:                }
190:
191:                return feedItems;
192:            }
193:
194:            /// <summary>
195:            /// fce prida do tabulky feeditem priznak precteni item polozky
196:            /// </summary>
197:            /// <param name="feedItem">id item polozky, ktera je
auktualizovana</param>
198:            public void IsReadFeedItem(FeedItem feedItem)
199:            {
200:                SqlCeCommand sComUpd1;
201:                sComUpd1 = new SqlCeCommand("UPDATE feeditem SET isread = 1 WHERE
title = @title AND feedId = @feedId", sCon);
202:
203:                sComUpd1.Parameters.Add("@title", SqlDbType.NVarChar).Value =
feedItem.Title;
204:                sComUpd1.Parameters.Add("@feedId", SqlDbType.Int).Value =
feedItem.FeedId;
205:                sComUpd1.ExecuteNonQuery();
206:            }
207:
208:            /// <summary>
209:            /// fce ulozi do tabulky feeditem item polozky z feedu(channel)
210:            /// </summary>
211:            /// <param name="feed">feed obsahuje polozky item</param>
```

```
212:          public void SaveFeedItem(Feed feed)
213:          {
214:              SqlCeCommand sComIns1, sComDel1;
215:
216:              sComDel1 = new SqlCeCommand("DELETE FROM feeditem WHERE feedId =
" + feed.Id, sCon);
217:              sComDel1.ExecuteNonQuery();
218:
219:              foreach (FeedItem item in feed.FeedItems)
220:              {
221:                  sComIns1 = new SqlCeCommand("INSERT INTO feeditem (title,
link, description, author, pubDate, isRead, feedId, warning) VALUES(@title, @link,
@description, @author, @pubDate, @isRead, @feedId, @warning)", sCon);
222:                  //sComIns1 = new SqlCeCommand("INSERT INTO feeditem (title,
link, description) VALUES(@title, @link, @description)", sCon);
223:
224:                  sComIns1.Parameters.Add("@title", SqlDbType.NVarChar).Value =
item.Title;
225:                  sComIns1.Parameters.Add("@link", SqlDbType.NVarChar).Value =
item.Link;
226:                  sComIns1.Parameters.Add("@description",
SqlDbType.NVarChar).Value = item.Description;
227:                  sComIns1.Parameters.Add("@author", SqlDbType.NVarChar).Value
= "";//item.Author;
228:                  sComIns1.Parameters.Add("@pubDate", SqlDbType.DateTime).Value
= item.PubDate;
229:                  sComIns1.Parameters.Add("@isRead", SqlDbType.Int).Value =
item.IsRead;
230:                  sComIns1.Parameters.Add("@feedId", SqlDbType.Int).Value =
feed.Id;
231:                  sComIns1.Parameters.Add("@warning", SqlDbType.Int).Value =
item.Warning;
232:
233:                  sComIns1.ExecuteNonQuery();
234:              }
235:          }
236:
237:          /// <summary>
238:          /// fce odstrani feed(channel) z tabulek feedstore a feeditem
239:          /// </summary>
240:          /// <param name="feedId">parametr obsahuje id feedu</param>
241:          /// <returns>vraci pocet ovlivnenych radku v tabulce</returns>
242:          public int DeleteFeed(int feedId)
243:          {
244:              SqlCeCommand sComDel1;
245:
246:              //smazeme napred vsechny feeditem polozky
247:              sComDel1 = new SqlCeCommand("DELETE FROM feeditem WHERE feedId =
" + feedId, sCon);
248:              sComDel1.ExecuteNonQuery();
249:
250:              //smazeme feed
251:              sComDel1 = new SqlCeCommand("DELETE FROM feedstore WHERE id = " +
feedId, sCon);
252:              int result = sComDel1.ExecuteNonQuery();
253:
254:              return result;
255:          }
256:
257:          /// <summary>
258:          /// fce smaze zaznam vybrane katekorie z tabulky category
259:          /// </summary>
260:          /// <param name="catId">obsahuje id kategorie</param>
```

```
261:          /// <returns>pocet ovlivnenych zaznamu</returns>
262:          public int DeleteCategory(int catId)
263:          {
264:              SqlCeCommand sComDel1 = new SqlCeCommand("DELETE FROM category
WHERE catId = " + catId, sCon);
265:              int result = sComDel1.ExecuteNonQuery();
266:
267:              return result;
268:          }
269:
270:          /// <summary>
271:          /// fce prida retezec (klicove slovo pro vyhledavani) do tabulky
warnings
272:          /// </summary>
273:          /// <param name="feedId">id feedu pro ktery bude retezec
uplatnovan</param>
274:          /// <param name="pattern">vzorek</param>
275:          /// <returns>vraci true pokud zaznam jiz existuje jinak
false</returns>
276:          public bool AddPattern(int feedId, string pattern)
277:          {
278:              SqlCeCommand sComIns1, sComSel1;
279:              SqlCeDataReader reader1;
280:              bool exist;
281:
282:              sComSel1 = new SqlCeCommand("SELECT feedId FROM warnings WHERE
feedId = " + feedId + " AND  pattern = '" + pattern + "'", sCon);
283:              reader1 = sComSel1.ExecuteReader();
284:
285:              exist = reader1.Read();
286:
287:              //pokud neexistuje zadany retezec a dany feedId
288:              if (!exist)
289:              {
290:                  sComIns1 = new SqlCeCommand("INSERT INTO warnings (feedId,
pattern) VALUES(" + feedId + ", '" + pattern + "')", sCon);
291:                  //vlozi novy zaznam do kategorie
292:                  sComIns1.ExecuteNonQuery();
293:             }
294:              return exist;
295:          }
296:
297:          /// <summary>
298:          /// fce nacte z tabulky warnings vzorky (klicova slova) pro
vyhledavani, ktere jsou pridany ke konkretnimu feedu(channel)
299:          /// </summary>
300:          /// <param name="feedId">id feedu</param>
301:          /// <returns>vraci ArrayLis bud naplneny stringem(vzorky), nebo
prazdny ArrayList</returns>
302:          public ArrayList LoadPatterns(int feedId)
303:          {
304:              ArrayList warnings = new ArrayList();
305:
306:              SqlCeCommand sComSel1 = new SqlCeCommand("SELECT pattern FROM
warnings WHERE feedId = " + feedId + "", sCon);
307:              SqlCeDataReader reader1 = sComSel1.ExecuteReader();
308:
309:              while (reader1.Read())
310:              {
311:                  warnings.Add(reader1["pattern"].ToString());
312:              }
313:
314:              return warnings;
```

```
315:          }
316:
317:          /// <summary>
318:          /// fce smaze vzorek z tabulky warnings
319:          /// </summary>
320:          /// <param name="feedId">id feedu(channel)</param>
321:          /// <param name="pattern">vzorek - string</param>
322:          public void DeletePattern(int feedId, string pattern)
323:          {
324:              SqlCeCommand sComDel = new SqlCeCommand("DELETE FROM warnings
WHERE feedId = " + feedId + " AND pattern = '" + pattern + "'", sCon);
325:              sComDel.ExecuteNonQuery();
326:          }
327:      }
328:  }
```

Třída Feeds

```
1:  using System.Collections.Generic;
 2:
 3:  ///<summary>
 4:  /// Trida vytvori objekt udrzujici reference na vsechny objekty:
List<Category>->Feeds->FeedItems
 5:  ///</summary>
 6:  namespace SysInfoRssReader
 7:  {
 8:      public class Feeds
 9:      {
10:          private List<Category> _feeds;
11:
12:          public Feeds()
13:          {
14:              _feeds = new List<Category>();
15:          }
16:
17:          public void AddCategory(Category category)
18:          {
19:              _feeds.Add(category);
20:          }
21:
22:          public List<Category> CategoryFeeds
23:          {
24:              get { return _feeds; }
25:
26:          }
27:      }
28:  }
```

Třída Category

```
1:  using System.Collections.Generic;
 2:
 3:  ///<summary>
 4:  /// Trida reprezentuje jednu kategorie, ktera odkazuje na referenci seznamu
feedu
 5:  ///</summary
 6:
 7:  namespace SysInfoRssReader
 8:  {
 9:      public class Category
10:      {
11:          private List<Feed> catFeed;
12:
```

```
13:          public Category(int catId, string catName)
14:          {
15:              catFeed = new List<Feed>();
16:              CatName = catName;
17:              CatId = catId;
18:          }
19:
20:          public int CatId { get; set; }
21:          public string CatName { get; set; }
22:
23:          public List<Feed> CatFeed
24:          {
25:              get { return catFeed; }
26:          }
27:
28:          public void AddFeed(Feed feed)
29:          {
30:              catFeed.Add(feed);
31:          }
32:      }
33:  }
```

Třída Feed

```
1:  using System.Collections;
 2:  using System.Drawing;
 3:  using System.Collections.Generic;
 4:
 5:  ///<summary>
 6:  /// Trida reprezentuje jeden RSS channel
 7:  ///</summary>
 8:
 9:  namespace SysInfoRssReader
10:  {
11:      public class Feed
12:      {
13:          public Feed()
14:          {
15:
16:          }
17:
18:          public Feed(int id, string title, string link, string pubDate, Bitmap
icon, int category, int itemCount, List<FeedItem> feedItems, ArrayList warnings)
19:          {
20:              Id = id;
21:              Title = title;
22:              Link = link;
23:              Icon = icon;
24:              Category = category;
25:              PubDate = pubDate;
26:              ItemCount = itemCount;
27:              if (feedItems == null)
28:              {
29:                  feedItems = new List<FeedItem>();
30:              }
31:
32:              FeedItems = feedItems;
33:              Warnings = warnings;
34:          }
35:
36:          public int Id { get; set; }
37:          public string Title { get; set; }
38:          public string PubDate { get; set; }
```

```
39:          public string Link { get; set; }
40:          public Bitmap Icon { get; set; }
41:          public string IconLink { get; set; }
42:          public int Category { get; set; }
43:          public int ItemCount { get; set; }
44:          public List<FeedItem> FeedItems { get; set; }
45:          public ArrayList Warnings { get; set; }
46:      }
47:  }
```

Třída FeedItem

```
1:  using System;
 2:
 3:  ///<summary>
 4:  /// Trida predstavuje informace o jedne item polozce z feedu
 5:  ///</summary>
 6:
 7:  namespace SysInfoRssReader
 8:  {
 9:      public class FeedItem
10:      {
11:
12:          public FeedItem()
13:          {
14:
15:          }
16:
17:          public FeedItem(string title, string link, string description, int
isRead, int warning)
18:          {
19:              Title = title;
20:              Link = link;
21:              Description = description;
22:              IsRead = isRead;
23:              Warning = warning;
24:          }
25:
26:          public string Title { get; set;}
27:          public string Link { get; set; }
28:          public string Description { get; set; }
29:          public string Author { get; set; }
30:          public DateTime PubDate { get; set; }
31:          public int IsRead { get; set; }
32:          public int FeedId { get; set; }
33:          public int Warning { get; set; }
34:      }
35:  }
```

Třída FeedsForm

```
  1:  using System;
  2:  using System.ComponentModel;
  3:  using System.Drawing;
  4:  using System.Windows.Forms;
  5:  using SysInfoRssReader.Properties;
  6:
  7:  ///<summary>
  8:  /// Formular zobrazuje kategorie -> feedy(channel), obsahuje objekt vybraneho
feedu
  9:  ///</summary>
 10:
 11:  namespace SysInfoRssReader
```

```
 12:  {
 13:      public partial class FeedsForm : Form
 14:      {
 15:          private readonly Program _program;
 16:          private Label _label;
 17:          private PictureBox _pictureBox;
 18:          private Panel _panel;
 19:          private LinkLabel _linkLabel;
 20:          private readonly ComponentResourceManager _resources;
 21:          private Feed _currentFeed;
 22:          private int _startPoint;
 23:
 24:          public FeedsForm(Program program)
 25:          {
 26:              InitializeComponent();
 27:              _resources = new ComponentResourceManager(typeof(Resources));
 28:              _program = program;
 29:
 30:              if (_program.FeedForm == null)
 31:              {
 32:                  _program.FeedForm = new FeedForm(_program);//pokud formular
FeedForm neni inicializovany
 33:              }
 34:          }
 35:
 36:          private void menuItem1_Click(object sender, EventArgs e)
 37:          {
 38:              PictureBoxClick(sender, e);
 39:          }
 40:
 41:          //aktualne vybrany feed
 42:          public Feed GetCurrentFeed
 43:          {
 44:              get { return _currentFeed; }
 45:          }
 46:
 47:          private void FeedsForm_Load(object sender, EventArgs e)
 48:          {
 49:              AutoScaleDimensions = new SizeF(96F, 96F);
 50:              AutoScaleMode = AutoScaleMode.Dpi;
 51:              AutoScroll = false;
 52:
 53:              //kontrola zdali jsou naplneny patricne objekty daty
 54:              if (_program.GetMainForm.GetFeeds.CategoryFeeds.Count == 0)
 55:              {
 56:
_program.GetMainForm.GetFeedStore.LoadFeedsList(_program.GetMainForm.GetFeeds);
 57:              }
 58:
 59:              _startPoint = 0;
 60:              Controls.Clear();
 61:
 62:              DrawHomeBtn(_startPoint);//btn home
 63:
 64:              _startPoint += 40;
 65:              DrawSilverLine(_startPoint);//line delimitter
 66:
 67:              _startPoint += 2;
 68:              DrawPanel(_startPoint);//main panel
 69:          }
 70:
 71:          //_panel na ktery se vykresluji kategorie, kanaly
 72:          private void DrawPanel(int startPoint)
```

```
 73:          {
 74:              HScrollBar hsb = new HScrollBar();
 75:              _panel = new Panel();
 76:              //_panel.Dock = System.Windows.Forms.DockStyle.Left;
 77:              _panel.Location = new Point(0, startPoint);
 78:              _panel.Name = "_panel";
 79:              _panel.Size = new Size(240, ClientRectangle.Height - startPoint +
hsb.Height);//Screen.PrimaryScreen.WorkingArea.Height
 80:              _panel.AutoScroll = true;
 81:              _panel.Anchor = AnchorStyles.Left;
 82:
 83:              Controls.Add(_panel);
 84:          }
 85:
 86:          //vykresleni kategorii
 87:          public void DrawCatLabel(string str, int startPoint)
 88:          {
 89:              _label = new Label();
 90:              _label.Location = new Point(2, startPoint);
 91:              _label.Font = new Font("Tahoma", 10F, FontStyle.Bold);
 92:              _label.ForeColor = Color.Black;
 93:              _label.Name = str;
 94:              _label.Size = new Size(236, 20);
 95:              _label.TextAlign = ContentAlignment.TopLeft;
 96:              _label.Text = Utils.CropText(str, 30);
 97:
 98:              _panel.Controls.Add(_label);
 99:          }
100:
101:          //linklabel - vykresleni jednotlivych feedu, ktere sepridaji na prvek
_panel
102:          public void DrawFeedLinkLabel(Feed feed, int startPoint)
103:          {
104:              _linkLabel = new LinkLabel();
105:              _linkLabel.Font = new Font("Tahoma", 9F, FontStyle.Regular);
106:              _linkLabel.ForeColor = _program.GetMainForm.GetTextColor;
107:              _linkLabel.Location = new Point(4, startPoint);
108:              _linkLabel.Size = new Size(220, 38);
109:              _linkLabel.Text = Utils.CropText(feed.Title, 70);//nazev feedu
110:              _linkLabel.Click += LinkLabelClick;
111:              _linkLabel.BackColor = _program.GetMainForm.GetBackgroundColor;
112:              _linkLabel.TextAlign = ContentAlignment.TopLeft;
113:              _linkLabel.TabStop = false;
114:              _linkLabel.Tag = feed;
115:
116:              _panel.Controls.Add(_linkLabel);
117:          }
118:
119:          //ImageBox - linka seda
120:          private void DrawSilverLine(int startPoint)
121:          {
122:              _pictureBox = new PictureBox();
123:              //_pictureBox.Dock = System.Windows.Forms.DockStyle.Top;
124:              _pictureBox.Image =
((Image)(_resources.GetObject("SilverLine")));
125:              _pictureBox.Location = new Point(0, startPoint);
126:              _pictureBox.Name = "DrawSilverLine";
127:              _pictureBox.Size = new Size(240, 2);
128:
129:              Controls.Add(_pictureBox);
130:          }
131:
132:          //ImageBox - linka seda, pro _panel
```

```
133:          private void DrawSilverLineOnPanel(int startPoint)
134:          {
135:              _pictureBox = new PictureBox();
136:              //_pictureBox.Dock = System.Windows.Forms.DockStyle.Top;
137:              _pictureBox.Image =
((Image)(_resources.GetObject("SilverLine")));
138:              _pictureBox.Location = new Point(0, startPoint);
139:              _pictureBox.Name = "DrawSilverLine";
140:              _pictureBox.Size = new Size(240, 2);
141:              //_pictureBox.Anchor = AnchorStyles.Left;
142:
143:              _panel.Controls.Add(_pictureBox);
144:          }
145:
146:          //ImageBox - linka bila
147:          private void DrawWhiteLine(int startPoint)
148:          {
149:              _pictureBox = new PictureBox();
150:              _pictureBox.Image = ((Image)(_resources.GetObject("WhiteLine")));
151:              _pictureBox.Location = new Point(0, startPoint);
152:              _pictureBox.Name = "DrawWhiteLine";
153:              _pictureBox.Size = new Size(240, 2);
154:
155:              _panel.Controls.Add(_pictureBox);
156:          }
157:
158:          //ImageBox - btn home
159:          private void DrawHomeBtn(int startPoint)
160:          {
161:              _pictureBox = new PictureBox();
162:              //_pictureBox.Dock = System.Windows.Forms.DockStyle.Bottom;
163:              _pictureBox.Image = ((Image)(_resources.GetObject("GoHomeBtn")));
164:              _pictureBox.Location = new Point(0, startPoint);
165:              _pictureBox.Name = "DrawHomeBtn";
166:              _pictureBox.Size = new Size(240, 40);
167:              _pictureBox.Click += PictureBoxClick;
168:
169:              Controls.Add(_pictureBox);
170:          }
171:
172:          //po kliknuti na btn home
173:          private void PictureBoxClick(object sender, EventArgs e)
174:          {
175:              _program.GetMainForm.Show();//navrat na main screen
176:          }
177:
178:          /// <summary>
179:          /// fce aktualizuje data, ktera jsou vykreslena na panel pred
zobrazenim formulare
180:          /// </summary>
181:          public void UpdatePanelData()
182:          {
183:              _panel.Controls.Clear();
184:
185:              int startPoint = 0;
186:              //pro vsechny kategorie vykresli kategori + feedy
187:              foreach (var catItem in
_program.GetMainForm.GetFeeds.CategoryFeeds)
188:              {
189:                  if (startPoint != 0)
190:                  {
191:                      DrawSilverLineOnPanel(startPoint);//oddeli carou prvky od
sebe - kategorie jednotlivych feedu
```

```
192:                     startPoint += 2;
193:                 }
194:
195:                 DrawCatLabel(catItem.CatName, startPoint);//vykresli
kategorie
196:                 startPoint += 20;
197:
198:                 foreach (var feedItem in catItem.CatFeed)
199:                 {
200:                     DrawFeedLinkLabel(feedItem, startPoint);//vykresli feed
201:                     startPoint += 38;
202:                     DrawWhiteLine(startPoint);
203:                     startPoint += 2;
204:                 }
205:             }
206:          }
207:
208:          /// <summary>
209:          /// fce nastavi vybrany feed(channel), ktery bude pouzit dale pro
zpracovani
210:          /// </summary>
211:          /// <param name="sender"></param>
212:          /// <param name="e"></param>
213:          private void LinkLabelClick(object sender, EventArgs e)
214:          {
215:              _currentFeed = (Feed)((LinkLabel) sender).Tag; //tag obsahuje
vybrany feed
216:
217:              _program.FeedForm.Show();
218:              _program.FeedForm.UpdatePaneldata();
219:          }
220:      }
221:  }
```

Dokument nápovědy aplikace SysInfoRssReader

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 2:  <html>
 3:      <head>
 4:              <title>Nápověda k aplikaci SysInfoRssReader</title>
 5:      </head>
 6:      <body>
 7:      <!-- PegHelp --><hr />
 8:          <a name="obsah" />
 9:          <h1>Obsah:</h1>
10:          <ul>
11:              <li><a href="SysInfoRssReaderHelp.htm#add">Add - přidat
kanál</a></li>
12:              <li><a href="SysInfoRssReaderHelp.htm#remove">Remove - odebrat
kanál/kategorii</a></li>
13:              <li><a href="SysInfoRssReaderHelp.htm#update">Update - aktualizace
dat</a></li>
14:              <li><a href="SysInfoRssReaderHelp.htm#feeds">Feeds - zobrazení
kanalů</a></li>
15:              <li><a href="SysInfoRssReaderHelp.htm#addwarnings">AddWarnings -
přidání klíčového slova pro vyhledávaní</a></li>
16:              <li><a href="SysInfoRssReaderHelp.htm#delwarnings">DelWarnings -
zobrazení kanalů</a></li>
17:              <li><a href="SysInfoRssReaderHelp.htm#about">About - o
aplikaci</a></li>
18:          </ul>
19:
20:          <!-- PegHelp --><hr />
```

```
21:          <a name="add" />
22:          <h1>Přidání kanálu</h1>
23:          <p>Formulář slouží pro přidání kanálu a kategorie do aplikace. Po
vyplnění údajů a kliknutí na tlačítko přidat, se stáhne z internetu název kanálu a
obrázek.
24:          Pokud není vyplněn název kanálu, tak se použije ten stažený. Obrázek
kanálu je použit, jen pokud má stejný poměr stran.</p>
25:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
26:
27:
28:          <!-- PegHelp --><hr />
29:          <a name="remove" />
30:          <h1>Odebrání kanálu</h1>
31:          <p>Pro odebrání slouží checkboxy. Aplikace po kliknutí na tlačítko
Remove odebre vybrané kanály. Pokud v kategorii neexistuje žádný kanál, je
kategorie odebrána taktéž.</p>
32:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
33:
34:
35:          <!-- PegHelp --><hr />
36:          <a name="update" />
37:          <h1>Aktualizace dat kanálů</h1>
38:          <p>Po kliknutí na tlačítko Update jsou stahována, pársovana data pro
kanály, které jsou zadány v aplikaci. Pokud kanál obsahuje klíčová slova, tak jsou
vyhledány v textu novinky.
39:          Formulář InfoForm zobrazuje průběh stahování a pársování dat. Barva
textu upozorňuje na prováděnou akci. Černá - neaktivní stahování, červená -
stahuje/pársuje data,
40:          zelená - spracováni dat je dokončeno.</p>
41:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
42:
43:
44:          <!-- PegHelp --><hr />
45:          <a name="feeds" />
46:          <h1>Zobrazení kanálu</h1>
47:          <p>Po kliknutí na tlačítko Feeds jsou zobrazeny kategorie/kanály.
Kategorie s kanály jsou od sebe vizuálně odděleny. Po kliknutí na položku kanálu je
vybrán tento kanál a jsou zobrazeny
48:           jeho aktuality. Pokud kanál obsahuje aktualitu, se sledovaným klíčovým
slovem, tak je titulek kanálu podbarven oranžovou barvou. Při výběru aktuality je
zobrazen její detail.</p>
49:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
50:
51:          <!-- PegHelp --><hr />
52:          <a name="addwarnings" />
53:          <h1>Sledování klíčových slov</h1>
54:          <p>V Menu - Settings - AddWarnings je umístěn formulář pro přidávaní
klíčových slov pro vyhledávaní v textu. Klíčová slova jsou přidružena ke
konkrétnímu kanálu.
55:          Pokud je v textu nalezeno klíčové slovo, tak je titulek novinky
podbarven oranžovou barvou.</p>
56:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
57:
58:
59:          <!-- PegHelp --><hr />
60:          <a name="delwarnings" />
61:          <h1>Odebrání klíčových slov</h1>
62:          <p>V Menu - Settings - DelWarnings je umístěn formulář pro odebrání
klíčových slov. Po vybrání kanálu jsou zobrazená klíčová slova, pokud je kanál
obsahuje. V opačném případě jsou
63:          ovládací prvky nedostupné.</p>
64:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
65:
```

```
66:
67:          <!-- PegHelp --><hr />
68:          <a name="about" />
69:          <h1>O aplikaci</h1>
70:          <p>Aplikace SysInfoRssReader slouží jako prototyp Rss čtečky se
zaměřením na sledování hodnot na monitorovaných systémech. Aplikace je napsána v
.NET CF 3.5 frameworku.
71:          Aplikace obsahuje implemntaci pro detekci připojení k síti, pársování
XML dat a SQL compact databázi ve verzi 3.5. Díky univerzálnosti formátu XML/RSS
2.0 je čtečka
72:          široce použitelná i pro zpracování dat z jiných zdrojů, které splňují
validnost RSS 2.0 formátu.</p>
73:          <p>Webové stránky autora: <a
href="https://feinuke.vsb.cz/stokpa/">https://feinuke.vsb.cz/stokpa/</a>.</p>
74:          <p><a href="SysInfoRssReaderHelp.htm#obsah">Zpět</a></p>
75:
76:          <!-- PegHelp -->
77:      </body>
78:  </html>
```